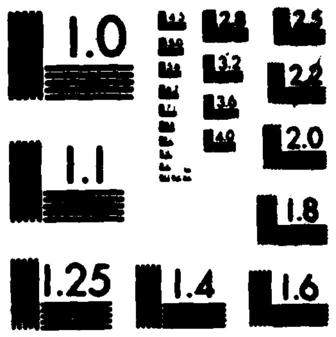
AD-A186 125 A GENERALIZATION OF ORDER STATISTIC FILTERS: THE ALPHA-TRIMMED LINEAR FILTER(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH D B RIDER 1987 1/2

UNCLASSIFIED AFIT/CI/NR-87-101T F/G 12/3 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A186 125 DTIC FILE COPY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFIT/CI/NR 87-101T	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Generalization of Order Statistic Filters: The a-Trimmed Linear Filter		5. TYPE OF REPORT & PERIOD COVERED THESIS/~~DISSERTATION~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Douglas B. Rider		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: University of Washington		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AFIT/NR WPAFB OH 45433-6583		12. REPORT DATE 1987
		13. NUMBER OF PAGES 172
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC ELECTE NOV 20 1987 S D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

D

18. SUPPLEMENTARY NOTES

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1

Lynn Wolaver 23 Sep 87

LYNN E. WOLAVER
Dean for Research and Professional Development
AFIT/NR

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

87 10 28 163

/101

A Generalization of Order Statistic Filters: The α–Trimmed Linear Filter

by

DOUGLAS B. RIDER

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

1987

Approved by [illegible]

Program Authorized to Offer Degree ELECTRICAL ENGINEERING

Date 27 MAY 87

Signature [illegible]

Date 27 MAY 87

TABLE OF CONTENTS

	Page
List of Figures	iv
List of Tables	vii
List of Symbols and Abbreviations	viii
Nomenclature	x
I. Introduction/Scope of This Research	1
II. The Order Statistic Filter (OSF)	4
L Filters	4
M Filters	5
III. The Generalized Order Statistic Filter (GOSF)	9
Definition	9
The α-Trimmed Linear Filter	11
Gain	11
IV. Performance Evaluation	14
Performance Measures	14
Frequency Selectivity	14
Transfer Functions	15
Outlier Resistance	17
Simulation Approach	19
V. Results	22
Frequency Selectivity	22
Effects of Filter Length	23
Effects of Filter Rolloff	32
Symmetry Properties	36

Page

Filter Gain 42

Overall Frequency Performance 44

Outlier Resistance 46

Visual Evaluation 47

NMSE Comparison 69

VI. Summary, Future Research and Conclusions 79

Summary 79

Future Research 80

Optimizing the Coefficients 80

IIR Filtering 80

Modified Trimmed Linear Filter (MTL) 81

Conclusions 82

Bibliography 84

Appendix A: Calculation of Transfer Functions 85

Random Number Generation 86

Window Selection 89

Histogram Distribution 90

Error Calculation 99

Appendix B: Computer Filter Model 105

Model Structure and Language Selection 106

Program Set Description 107

Appendix C: Set of Linear Filters Used 110

Appendix D: Program Listings 130

LIST OF FIGURES

Number Page

1. Ø(x) for a STM Filter 6
2. Generalized Order Statistic Filter 9
3. α–TL BPF2500s L31 Transfer Functions 25
4. α–TL BPF2500s L63 Transfer Functions 26
5. α–TL HPF3500 L31 Transfer Functions 27
6. α–TL HPF3500 L63 Transfer Functions 28
7. α–TL BPF2500s L31 Transfer Functions plus Ideal Frequency Response 30
8. α–TL BPF2500s L63 Transfer Functions plus Ideal Frequency Response 31
9. α–TL BPF2500 L31 Transfer Functions plus Ideal Frequency Response 34
10. α–TL BPF2500 L31 Transfer Functions 35
11. α–TL LPF300 L31 Transfer Functions plus Ideal Frequency Response 37
12. α–TL LPF300 L63 Transfer Functions plus Ideal Frequency Response 38
13. α–TL LPF1500 L15 Transfer Functions plus Ideal Frequency Response 40
14. α–TL HPF3500 L15 Transfer Functions plus Ideal Frequency Response 41
15. Gain Factor for α–TL BPF2500s 43
16. Gain Factor for α–TL BPF2500 44

Number Page

17. First Sidelobe Level of α–TL BPF2500s 45
18. SIG1G – signal 1 + noise ~N(0,0.2) 48
19. SIG1I – signal 1 + noise ~L(0,0.5) 49
20. SIG1IG – SIG1I + noise ~N(0,0.1) 50
21. SIG1G Filtered by Mean 9 52
22. SIG1G Filtered by Median 9 53
23. SIG1G Filtered by α–TM 9T6 54
24. SIG1G Filtered by LPF300 L9 55
25. SIG1G Filtered by α–TL LPF300 9T6 56
26. SIG1G Filtered by LPF1500 L9 57
27. SIG1G Filtered by α–TL LPF1500 9T6 58
28. SIG1I Filtered by Mean 9 59
29. SIG1I Filtered by α–TM 9T6 60
30. SIG1I Filtered by Median 9 61
31. SIG1I Filtered by LPF300 L9 63
32. SIG1I Filtered by α–TL LPF300 9T6 64
33. SIG1I Filtered by LPF1500 L9 65
34. SIG1I Filtered by α–TL LPF1500 9T6 66
35. Results of SIG1IG Filtered by Median 9 and
α–TL LPF300 9T8 67
36. SIG1G Filtered by α–TM – NAE 71
37. SIG1G Filtered by α–TM – NMSE 71
38. SIG3G Filtered by α–TM – NMSE 72
39. SIG1G Filtered by α–TL – NMSE 73
40. SIG3G Filtered by α–TL – NMSE 73

Number Page

41. SIG3I Filtered by α–TM – NMSE 75
42. SIG3I Filtered by α–TL – NMSE 76
43. SIG2G Filtered by α–TL – NMSE 77
44. SIG2I Filtered by α–TL – NMSE 77
A1. Autocorrelation of 2048 Gaussian Samples 91
A2. Transfer Function of BPF2500s L31 (linear scale) 94
A3. Histogram of 500 Values of Sample 91 95
A4. Histogram of 500 Values of Sample 92 96
A5. Histogram of 500 Values of Sample 93 97
A6. Histogram of 500 Values of Sample 94 98
A7. 80% Confidence Interval for BPF2500s 15T2 102
A8. 80% Confidence Interval for BPF2500s 15T6 103

LIST OF TABLES

Number		Page
1.	LPF1500 L9 and HPF3500 L9 Coefficients	39
A1.	Random Number Transformation	89
C1.	Filter Coefficients for LPF300 L9	111
C2.	Filter Coefficients for LPF300 L15	112
C3.	Filter Coefficients for LPF300 L31	113
C4.	Filter Coefficients for LPF300 L63	114
C5.	Filter Coefficients for LPF1500 L9	115
C6.	Filter Coefficients for LPF1500 L31	116
C7.	Filter Coefficients for LPF1500 L63	117
C8.	Filter Coefficients for BPF2500s L9	118
C9.	Filter Coefficients for BPF2500s L15	119
C10.	Filter Coefficients for BPF2500s L31	120
C11.	Filter Coefficients for BPF2500s L63	121
C12.	Filter Coefficients for BPF2500 L9	122
C13.	Filter Coefficients for BPF2500 L15	123
C14.	Filter Coefficients for BPF2500 L31	124
C15.	Filter Coefficients for HPF2500 L31	125
C16.	Filter Coefficients for HPF3500 L9	126
C17.	Filter Coefficients for HPF3500 L15	127
C18.	Filter Coefficients for HPF3500 L31	128
C19.	Filter Coefficients for HPF3500 L63	129

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol	Definition
α	parameter to determine number of elements trimmed off of a filter
α–TL	α–trimmed linear
α–TM	α–trimmed mean
$\emptyset(x)$	continuous, odd, sign preserving function
a_n	time weighting coefficient in generalized order statistic filter model
b_n	rank weighting coefficient in generalized order statistic filter
BPF	band pass filter
FIR	finite impulse response
g	gain factor
GOSF	generalized order statistic filter
$h(n)$	filter impulse response
H	transfer function
HPF	high pass filter
I	impulsive (Laplacian)
$i(n)$	ideal signal
IIR	infinite impulse response
L	length of filter
L	Laplacian
LPF	low pass filter
MTL	modified trimmed linear
MTM	modified trimmed mean

Symbol	Definition
N	length of filter; length of signal sequence
N	normal (Gaussian)
NAE	normalized average error
NMSE	normalized mean square error
OSF	order statistic filter
P_{xx}	auto spectral density
P_{yx}	cross power spectral density
T	trimmed; number of elements trimmed
T_{yx}	transfer function
x; $x(n)$	input signal
y; $y(n)$	output signal
z; $z(n)$	output signal

I. INTRODUCTION/SCOPE OF THIS RESEARCH

With the increase in use of digital communications has come a renewed interest in nonlinear filtering techniques. In some signal processing applications the suppression of noise on a discrete-time signal sequence can not be adequately accomplished using linear filtering. In order to preserve sharp edges in a signal while still smoothing noise and removing impulsive components requires some type of nonlinear or adaptive filtering.

The most common nonlinear smoother to be investigated is the median filter. In this filter the median is taken from a finite length window of values surrounding the input value. This scheme has been shown to preserve edges in signals with a minimum of distortion and is resistant to outliers in the data. However, it often does not provide adequate smoothing of noise that is not by nature impulsive. Yet the desirable properties of median filtering make it worth investigating.

Several methods have been investigated to overcome the limitations of median filters while retaining some of its advantageous properties. These methods have sought to retain the impulse rejection and edge preservation properties of the median filter while improving the Gaussian noise suppression. Several of the more prominent filters are discussed in Section II. This section shows the progression of hybrid filter designs combining the desirable properties of linear and nonlinear filters.

Yet this progression, while impressive and important, does not

/0/

NOMENCLATURE

Throughout the course of the simulation several different methods were used to describe the various filters and filter operations under consideration. First, median and mean refer to the two filtering operations denoted by those words: median and mean filters. However, the word median is also used in several titles to denote transfer functions that were calculated using the median method described in Appendix A. Transfer functions not calculated using the median method were calculated by averaging.

The α–trimmed mean and α–trimmed linear filters are denoted by either a–TM and a–TL or α–TM and α–TL. The size and number of elements trimmed is also often specified in most titles. An α–trimmed mean filter of length 9 with 4 elements trimmed is specified by α–TM 9T4. Note that α–TM 9T0 is the same thing as mean 9 or mean L9 and that α–TM 9T8 is the same thing as median 9 or median L9.

The α–trimmed linear filters are specified in the same manner with an additional designation to show which linear filter is being used. The linear filter designation standing alone refers to the untrimmed linear filter. For example, LPF1500 L31 refers to a low pass filter of 31 elements with a cut off frequency of 1500Hz (based on a 10kHz sampling rate). Equivalent designations for this same filter are LPF.15 L9 and LPF1500 9. The numbers following the three letter filter designation—LPF, BPF, or HPF—indicate a frequency associated with that filter. For the low pass and high

87 10 22 163

pass filters this number represents the frequency in the center of the transition band, the cut off frequency. For the band pass filter the number represents the frequency in the center of the pass band. The frequency may be specified as a normalized frequency from 0 to 0.5 or as a frequency in Hertz based on a 10kHz sampling rate. Designations for an α-trimmed linear filter look like α-TL BPF2500 31T2. This designation is for a 31 element band pass filter with a pass band centered at 2500Hz having 2 elements α-trimmed.

There are some special one letter designations. BPF2500s is used to identify the "skinny" BPF centered at 2500Hz with a sharper roll off than BPF2500. The following one letter designations refer to the filters opposite them:

X(tra low pass)	LPF300
L(ow pass)	LPF1500
B(and pass)	BPF2500s
F(at band pass)	BPF2500
H(igh pass)	HPF3500.

These designations were used when a condensed notation was necessary. They are often seen as F31T2 to designate α-TL BPF2500 31T2.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Ritcey for his assistance in making this thesis a reality. In addition, special thanks to Mr. Dave Fray and Mr. Ed Gulbransen without whom I would never have been able to tackle the computer problems that constantly impeded my path to this goal.

address a particularly important characteristic of linear filters: frequency selectivity. There are many applications in which the signal of interest is not obtained using a simple low pass filter—the median filter being a low pass operation—but rather a filter selecting some other frequency band. It is addressing these applications—matched filter banks and IF and RF filters, for example—to which this research effort is directed.

It is a relatively simple matter to design a linear filter to pass a certain band of frequencies. However, if the environment through which the signal is propagated happens to be impulsive rather than Gaussian, say under the polar ice cap as an extreme example, then a linear filter may not provide adequate smoothing of the noise. In this case it would be desirable to have some of the outlier resistance properties of the median filter in order to deal with the impulsive noise.

Section III describes the generalized order statistic filter (GOSF) model. An α-trimmed linear (α-TL) filter is defined using this model which is an attempt to meet the goal of a frequency selective filter having the outlier resistance properties of the nonlinear median filter. In this section the α-TL filter model is defined based on the logical progression of hybrid filters described in Section II. Several important considerations of this design are also discussed at the end of Section III.

In Section IV some performance measures are defined with which the α-TL filter will be evaluated. For this filter model the performance measures of interest are, of course, outlier resistance and frequency selectivity. Special attention is given to how the frequency performance of the nonlinear α–TL filter is assessed. A brief discussion of some of the important simulation parameters is also included.

The results of the simulation are presented in Section V. Several effects of the trimming process on the linear filter characteristics are discussed to provide insight into the filtering mechanism. More importantly, several specific effects of filter coefficient parameters on the performance of the α–TL filter are discussed in detail. This discussion centers on how varying certain filter design parameters such as the number of elements in the filter and the steepness of the designed filter rolloff seem to affect the performance of the α–trimmed filters in general.

The analysis of these effects is summarized in Section VI. The overall performance of the α–TL filter is assessed and possible avenues for improvement/future research on the trimmed linear filter are discussed.

II. THE ORDER STATISTIC FILTER

A. THE L FILTER

Bovik, Huang, and Munson introduced a generalization of the median filter which they called an order statistic filter (OSF). This filter uses some linear combination of the ordered data in a window around an input point to produce the output. With the proper choice of coefficients one may create a running mean filter (all the coefficients equal to $1/N$), a median filter, or something in between. In addition, a minimum or maximum filter is also a special case of the OSF.[1] Lee and Kassam noted that an OSF is the same thing as an L filter since an L filter is, *by definition*, a linear combination of order statistics.[2]

Lee and Kassam went on to develop a simple representation of the L filter called an α-trimmed mean (α-TM) filter. In this representation the filter has a requirement that all the weights in some central portion of the window be equal to a constant with the rest of the weights taken to be zero. The number of samples trimmed off each end of the window, T, is parameterized by α, $T = \lfloor \alpha(2N+1) \rfloor$, where $0 \leq \alpha \leq 0.5$ and $\lfloor X \rfloor$ is the largest integer less than or equal to X. This filter also has the two special cases of the running mean filter, $\alpha=0$, and the median filter, $\alpha=0.5$. The parameter α is chosen based on *a priori* knowledge of the noise distribution or by an adaptive scheme.[3]

The output y_k for this filter is given by

$$y_k = \sum_{j=T+1}^{2N+1-T} x^k_{(j)}/[2(N-T)+1], \qquad (1)$$

where $x^k_{(j)}$ is the jth order statistic of the window centered around the kth element in the signal. Lee and Kassam observed that the α-TM filter possesses a clear trade-off between the advantages of a mean and a median filter. As α approaches 0 or 0.5 the characteristics of the filter approaches a mean or median filter, respectively.[4]

B. M FILTER

Lee and Kassam also proposed an M filter with a more favorable combination of the two individual filters' characteristics. The output y_k of an M filter is defined as the solution to the equation

$$\sum_{i=k-N}^{k+N} \phi(x_i - y_k) = 0, \qquad (2)$$

where ϕ is some odd, continuous, and sign-preserving function. When ϕ is the linear function $\phi(x) = ax$, for a=constant, the M filter reduces to the running mean filter. On the other hand, the M filter's characteristics approach those of the median filter as $\varnothing(x) \rightarrow \mathrm{sgn}(x)$ under certain conditions.[5] They also defined a *standard type* M filter (STM filter) as a limiter type filter for which

$$\varnothing(x) = \begin{cases} g(p), & x > p \\ g(x), & |x| \leq p \\ -g(p), & x < -p \end{cases} \qquad (3)$$

and $g(x) = ax$ as shown in Figure 1.

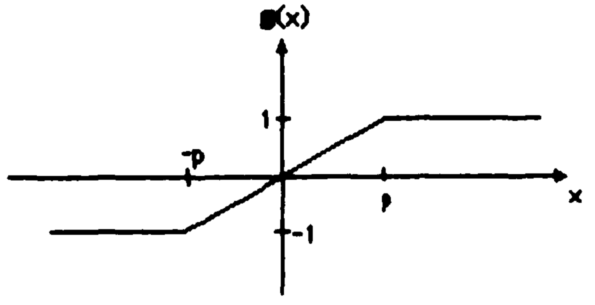


Figure 1: $\varnothing(x)$ for a STM filter[6]

The STM filter was found to have very favorable characteristics. Its window is somewhat data dependent so that it tends to treat as outliers those values which are very large or very small as compared to the median. This leads to the interpretation of an STM as a data dependent type of L filter that may have a nonsymmetric window. Lee and Kassam concluded that an STM tended to act as a running mean filter when neither edges nor impulsive noise were present and as a median filter over areas with edges. It combined the advantages of the two types of filters more favorably than an L filter. The major disadvantage of the STM is that it is difficult to evaluate the output of the filter. In their simulation, Lee and Kassam used the iterative Newton's method.[7]

Finally, Lee and Kassam noted that the main difference between an α-TM filter and an STM filter was that the number of samples trimmed from each end of the window of an STM was data dependent and not necessarily symmetric. This was the reason STM filters could outperform the simple α-TM filter. Applying this observation they came up with a modified trimmed mean (MTM) filter which is simple to implement and, in many cases, produces results at least as good as those obtained with an STM filter.[8]

The MTM filter first determines the sample median m_k inside its window and then chooses an interval $[m_k - q, m_k + q]$ using some preselected constant q. All data samples whose value lie outside the range are discarded and the average of the remaining values is taken as the output at sample k. This is a similar operation to an α-TM filter only for the MTM filter the range around the median is the determining factor in how many samples are discarded instead of the constant number of discarded samples determining the range of values to average in the α-TM filter. The MTM filter also differs from the STM filter in that the MTM filter may discard the values that lie outside the preselected range whereas the STM filter only limits these values to the range itself.[9]

References

[1] Alan C. Bovik, Thomas S. Huang and David C. Munson, Jr., "A Generalization of Median Filtering Using Linear Combinations of Order Statistics," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-31, No. 6, Dec 1983, 1343.

[2] Yong Hoon Lee and Saleem A. Kassam, "Generalized Median Filtering and Related Nonlinear Filtering Techniques," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 3, Jun 1985, 673.

[3] Lee and Kassam, 674.

[4] Lee and Kassam, 674.

[5] Lee and Kassam, 675.

[6] Lee and Kassam, 676.

[7] Lee and Kassam, 677-8.

[8] Lee and Kassam, 678.

[9] Lee and Kassam, 678.

III. THE GENERALIZED ORDER STATISTIC FILTER

A. DEFINITION

Seeing these developments and the improvements that they allow makes one wonder at the possibilities of merging the capabilities of linear and nonlinear filtering techniques. Using the notation that has been discussed up to this point, we will look at the possibilities of a filtering scheme proposed by Ritcey that may be called a generalized order statistic filter (GOSF). The scheme works like this: a signal is passed through a set of time weighting coefficients designed for some particular frequency response. But before the results are summed, the values are ordered and trimmed according to some scheme. This is easier to visualize graphically as shown in Figure 2.

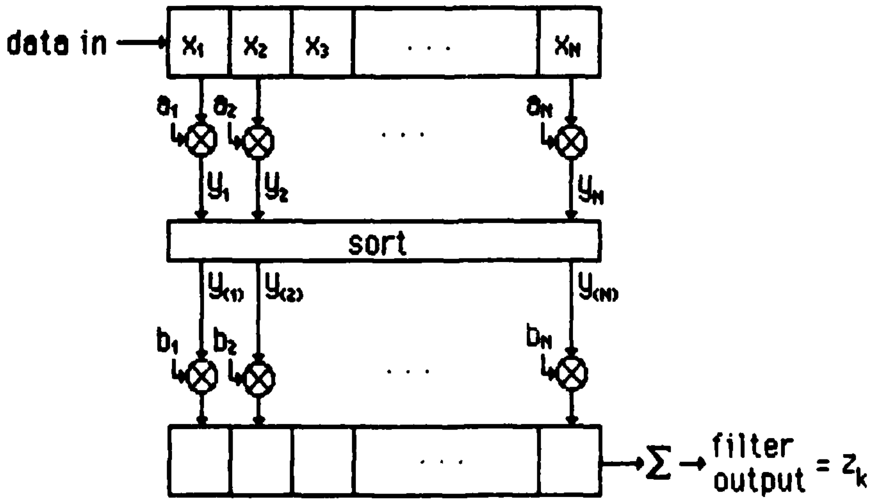


Figure 2: Generalized Order Statistic Filter

A notable feature of this system is that many other filters are special cases of this generalized model. As with the L filter in general and the α–TM filter specifically, the running mean and median filters are easily and obviously attained by choosing the proper coefficients for this model. In addition, one notes that the L filter is also a subset of this general model if we choose all of the a_n's to be unity. Finally, one can also obtain an MTM filter by providing a mechanism to check the signal values as they come out of the sorter and choose the b_n coefficients based on the median and a preselected range parameter.

The output z_k is given by

$$z_k = (1/g) \sum_{j=T+1}^{2N+1-T} b_j y^k{}_{(j)} \tag{4}$$

where $y^k{}_{(j)}$ = ordered$[y_i]$ centered around x_k and

$$y_i = a_i x_i \; ; \qquad i = 1, 2, \ldots N \tag{5}$$

and g is the gain factor to be discussed later.

B. THE α-TRIMMED LINEAR FILTER

The α-trimmed linear (α-TL) filter is obtained from the generalized order statistic filter by using a set of linear filter coefficients as the time weighting coefficients. The signal enters a window and is weighted by the a_n's just as if it were going to be linearly filtered. Only before the weighted values are summed to produce the filter output as in a linear filter operation, they are sorted and trimmed (the b_n's being restricted to zero or one). Thus we have a sort of trimmed "linear" filter similar to the α-trimmed mean filter. The main difference between the two filters is that the α-TM filter has a_n's that are a constant equal to $1/(N-T)$ while the α-TL filter has a set of FIR coefficients for the a_n's which are modified by a constant, the gain factor g.

C. GAIN

A serious consideration in this model is the gain of the system. Without some modification, if a set of linear FIR filter coefficients that sum to a particular value (say unity for a low pass filter) are used and then some of the values are trimmed off, there would be some gain in the system that may make it difficult to see what is actually happening. To take into account the gain of the system we look at what happens to a constant signal passing through it. To overcome the attenuation the linear set of coefficients are ordered and those coefficients that would not be trimmed by the filter are summed. As an example, suppose a constant signal of unity was passed through a hypothetical seven element LPF whose coefficients were as follows:

$$a_n\text{'s} = \frac{-1}{16}, \frac{2}{16}, \frac{4}{16}, \frac{6}{16}, \frac{4}{16}, \frac{2}{16}, \frac{-1}{16}$$

$$\text{where} \quad \sum a_n = 1.$$

Ordering these we would get -1, -1, 2, 2, 4, 4, and 6 sixteenths. If, in our filtering scheme, we happened to trim off the largest and smallest values and summed the result we would get 11/16 as the filtered output of the constant unity input. Thus our gain needs to be 16/11 which corresponds to summing the a_n's that are not trimmed and dividing the filter output by this amount. The gain factor is given by

$$g = \sum a_n - \sum_{i=1}^{T/2} (a_{(i)} + a_{(N+1-i)}) \qquad (6)$$

where the $a_{(i)}$ = ordered$[a_n]$ and T is the number of elements being trimmed (always even). In general, however, a filter

IV. PERFORMANCE EVALUATION

A. PERFORMANCE MEASURES

In evaluating this proposed filtering scheme concentration will be focused on two major filter characteristics. Our primary interest will be trying to maintain the frequency selectivity of a linear filter (i.e. being able to create a filter to pass a certain frequency band) while adding in the inherently nonlinear characteristic of outlier resistance. These two capabilities do not exist simultaneously in any of the filters discussed previously. In all of the filter models discussed up to this point the application was assumed to be that of a low pass filter. What if the application calls for a selection of some other frequency band as the signal of interest? In this case a linear filter is needed, but we already know that a linear filter is highly susceptible to the effects of outliers on the input signal. Is there a filter design that will allow a frequency selection of other than a low pass filter while providing improved performance in the presence of impulsive noise? We are going to evaluate the proposed α-trimmed linear filter in regards to these two performance measures: outlier resistance and frequency selectivity.

FREQUENCY SELECTIVITY

The evaluation of the frequency selectivity of a filter is going to be based on how well the modified filter's frequency characteristics match that of the unmodified linear filter. This criteria was chosen because it is relatively simple to evaluate. It does not require any modification of the filter coefficients. We can just compare how well the frequency response of the modified filter compares to that of the unmodified filter. This allows an evaluation of how the ordering and trimming of the α–trimmed linear filter affects the frequency characteristics of the original linear filter.

We could have chosen to set a particular frequency characteristic as the desired response and, through some iterative procedure, changed the coefficients of the filter adaptively to best imitate this desired response. However, due to the nonlinear characteristics of the α–TL filter this could prove difficult to do. Therefore, this possibility was not explored. This could be an area for further study.

In evaluating the frequency selectivity of the α–TL filter we will first compare visually the frequency response characteristics of the α–TL filter to those of the original linear filter. This may allow a quick insight into the actual physical process of the filter and may suggest ways to improve filter performance. Second, we will do an error analysis of the modified filter's frequency characteristics compared to those of the original linear filter. This will allow a quantitative evaluation of how much degradation in frequency performance is introduced by the α–trimming process.

TRANSFER FUNCTIONS

A problem encountered here is how to measure the frequency characteristics of a particular α–TL filter. The frequency response of an FIR filter is found simply by using an impulse as the input to the filter and taking the DFT of the output. This is the transfer function of the filter because the DFT of the input (an impulse) is unity at all frequencies. However, we cannot find the transfer function of the α-trimmed linear filter in the same manner because it is designed to reject impulses in the input signal. By using an impulse as the input into an α-TL filter, with even only two elements trimmed off, the output of the filter is going to be zero at all times.

To overcome this limitation in measuring the transfer function of the nonlinear filter, white Gaussian noise was used as the input to the filter. The transfer functions were then computed as follows. An input $x(n)$ of white Gaussian noise was filtered to get the output $y(n)$. The input and output sequences were both Fourier transformed to get $X(e^{j\omega})$ and $Y(e^{j\omega})$, respectively. The transfer function $H(e^{j\omega})$ was given by

$$|H(e^{j\omega})| = |Y(e^{j\omega})| \;/\; |X(e^{j\omega})|. \tag{7}$$

This calculation gives the proper magnitude of the transfer function which is more often calculated by

$$T_{yx} = P_{yx} \;/\; P_{xx} \tag{8}$$

where P_{yx} is the cross power spectral density of the output with the input (= $Y(e^{j\omega})X^*(e^{j\omega})$) and $\mathbf{P}_{xx}$ is the auto power spectral density of the input (= $X(e^{j\omega})X^*(e^{j\omega})$). This calculation preserves the phase information of the transfer function as well. However, we are only interested in the magnitude of the transfer function.

Experimentally it was found that by choosing a window function for the input signal that minimized leakage, the transfer function for the linear FIR filter calculated using white noise as the input was equal to the transfer function using an impulse to within the resolution of the computer. That is to say the error between the transfer functions calculated using the two different inputs, if any, could easily be attributed to round off error within the machine. The window function is discussed in more detail in Appendix A.

OUTLIER RESISTANCE

As was noted previously, FIR filters inherently perform poorly in the presence of impulsive noise on a signal. This is due to the effect a large impulse has on a linear filter. Being summed in each element as the entire window passes over it creates a bias in the output signal in the direction of the impulse. We would like to minimize the effect of an outlier by discarding it as in a median type filter. If this were possible then the effect of an impulse on the input signal would become very small. The only performance sacrifice would be, in effect, a shortening of the filter length by one element thus decreasing the Gaussian noise suppression capability of the original linear filter. While this is a simplification of the actual filtering process, it serves to illustrate the differences in effect an impulse has on a linear filter versus a nonlinear filter.

We can evaluate a filters performance in rejecting outliers in two ways. The simplest and most straightforward method is a visual measure of comparing graphs of test signals to see which filter produces the smoothest output with impulses present on the input signal. While this method may seem imprecise it does allow an immediate evaluation of a filter's performance upon a particular input signal and a quick comparison of different filters' performance. By comparing the input to the output at any given instant it also gives an insight into the filter's physical process. This gives an understanding of why a filter performs well in some situations and poorly in others and may suggest ways to modify a particular filter design to perform better in some respect.

The second method for evaluating performance would be to compute the error between the filter output and the uncorrupted input signal to try to numerically determine which filter best smooths noise. This would allow a direct comparison of different filters' performance on a particular set of input signals. With a proper choice of test signals we should be able to derive some insight into how well a particular filter design will perform under a given set of input conditions. In order to find out what trade-offs are being made we will evaluate the performance of several filter designs on several different input signals under three test conditions: 1) the signal corrupted by Gaussian noise only; 2) the signal corrupted by impulsive noise only; and 3) the signal corrupted by both Gaussian and impulsive noise. This will allow us to see if, by adding a capability to deal with outliers on the input signal, we are giving up too much Gaussian noise suppression.

B. SIMULATION APPROACH

There are several specific aspects of the simulation that are worth mentioning. First, the equations for the calculation of the transfer functions for the frequency selectivity tests, given in part A of this section, do not allow the transfer functions to be calculated deterministically. Since the inputs for these calculations were Gaussian noise, and therefore random, the transfer functions were found probabilistically. Five hundred trials of random noise were generated to calculate each transfer function. Each of these were sent through the filter separately, Fourier transformed separately and then the output transform magnitude was divided by the input transform magnitude to generate a single trial transfer function and stored point by point. When the transfer functions of all 500 trials had been calculated and stored, the results for each point were then ordered from smallest to largest. The median at each point was chosen to represent the transfer function of the α–TL at that frequency.

The median was used instead of the mean in determining the transfer function at each point for two reasons. First, the histogram of values at each point usually was quite skewed. Most of the values were lumped at a certain distance from zero (there could be no negative values) with an exponential distribution fading away from the main lobe. Second, and more importantly, due to the nonlinearity of the filtering there were individual values which were hundreds of times larger than most of the others. These individual values could affect the output at a single point by as much as 20% of the average value. For a complete discussion of the transfer function histograms see Appendix A.

Second, in evaluating the outlier resistance performance of the α–TL filter and comparing it to some of the other filters described previously it was important to develop a set of test signals that would allow observations on particular signal characteristics. This was basically narrowed to a comparison of signals with only smooth, "slow" variations—slow being relative to the frequency band of the filter—to signals with sharp edges. A suitable set of input signals was developed and included several signals of very smooth slow variations, a sine wave or slow ramp, for example, several signals of step functions and a couple of combination signals. The combination signals generally showed the most interesting results and are used for most of the examples shown in Chapter V.

Once this set was developed each signal was corrupted with noise. To each signal was added 1) Gaussian white noise only; 2) impulsive (Laplacian or double–exponential) white noise only; and 3) both Gaussian and impulsive additive noise. The purpose of these three testing schemes was to try to show that the α–trimmed mean and α–trimmed linear filters performed better than some of the other models in impulsive noise. The comparisons would allow us to see how much Gaussian noise suppression was being given up for this outlier rejection.

V. RESULTS

A. FREQUENCY SELECTIVITY

As one would probably expect, the frequency characteristics of the α–trimmed linear filter are similar to those of the corresponding linear filter. However, as one might also expect, the frequency performance is not as good. Naturally, trimming elements off of a set of coefficients will produce some side effects, a degradation of performance. This is especially pronounced in the α–TL filter for a couple of reasons.

First, we have a randomness associated with the nonlinearity of this particular trimming process. The element chosen to be trimmed, and thus the effect a particular coefficient of the linear filter has on the output at a given time, varies from point to point depending on the input data surrounding the point. The trimming takes on a data dependence. This inherent nonlinearity of the filter introduces a randomness that can only be expected to upset the balance of a set of linear coefficients.

Second, although the trimming is data dependent and therefore somewhat random, in this filter it is the largest and smallest (or rather the largest positive and largest negative) values that are trimmed. In the absence of large outliers on the input signal, we can expect that these values will correspond to some of the largest (in absolute value) filter coefficients. This means that some of the coefficients that would normally contribute the most to the output at a given point in the linear filter are the elements that are being trimmed.

These two observations help us understand some of the process that causes a degradation of performance. Experimentally, however, several interesting and perhaps not so intuitive things were discovered about what effects the α–trimming process has on a set of linear filter coefficients and the effects of certain filter coefficient parameters on the trimming process itself. Specifically there were three notable items: 1) the effects of the filter's length; 2) the effects of the designed linear filter's rolloff characteristics; 3) the symmetry properties maintained through the trimming; and 4) the DC gain problem the α–trimming process introduces. Each of these topics will be discussed in some detail.

EFFECTS OF FILTER LENGTH

As we know from linear filter design class, the more elements we have in a filter the better the frequency response we may obtain. More elements in the filter mean we can design a better attenuation in the stop band, a steeper rolloff in the transition band, a flatter response in the pass band or more often some combination of these design parameters. Thus, when we start trimming away some of the coefficients of a filter we can only expect that some or all of these basic parameters may suffer. In addition, if the trimming is done in a random or nonlinear fashion we can expect a certain amount of randomness or nonlinearity to be introduced into these filter parameters as well.

During a normal filter design, one of the major trade–offs encountered is the cost of increased complexity caused by a longer filter with the desired stop band attenuation and transition band rolloff. The importance of the flatness of the pass band and stop band are normally accounted for by the choice of the filter design—Butterworth, Chebyshev, Elliptical, etc.—and to a smaller extent by the length of the filter. Usually the driving factor in a filter design will be that the attenuation in the stop band meet some minimum requirement. This is balanced against the cost (complexity/length) of the filter.

An interesting effect occurs, however, when we start talking about an α-trimmed linear filter with a set of FIR coefficients. An attenuation limit is reached quite rapidly even though the filter is made longer! By comparing the graphs of BPF2500s for the lengths of 31 and 63 elements, Figures 3 and 4, and the graphs of HPF3500 for these same lengths, Figures 5 and 6, one can see that the attenuation does not get any better for the longer filter. In fact, if the graphs were superimposed it could be seen that the responses are nearly identical. This is especially true for the T=2 graphs. The attenuation is the same for all frequencies with only some minor discrepancies in the rolloff region. As T increases this variation in the transition band increases somewhat. The rolloff is slightly steeper for the longer filters as would be expected. In addition, the first hump is slightly narrower, a little higher, and rises more sharply for the longer filter. These facts all seem to be in line with what we would expect for a larger linear filter.

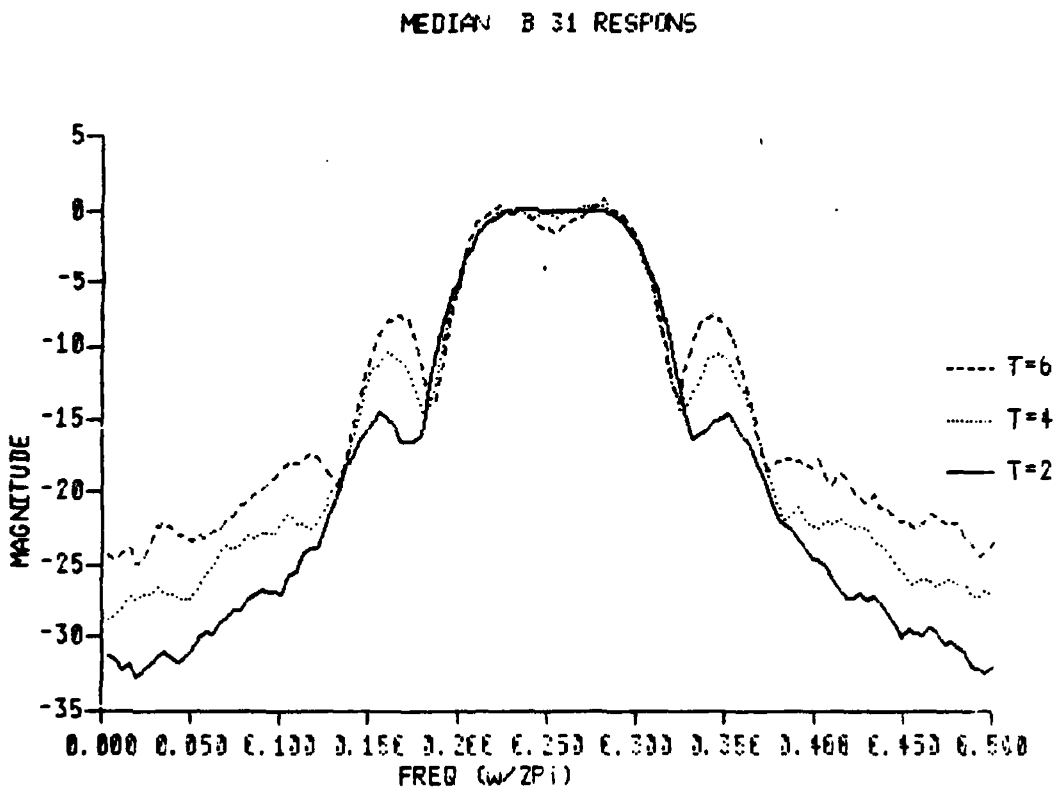


Figure 3 : α–TL BPF2500s L31 Transfer Functions

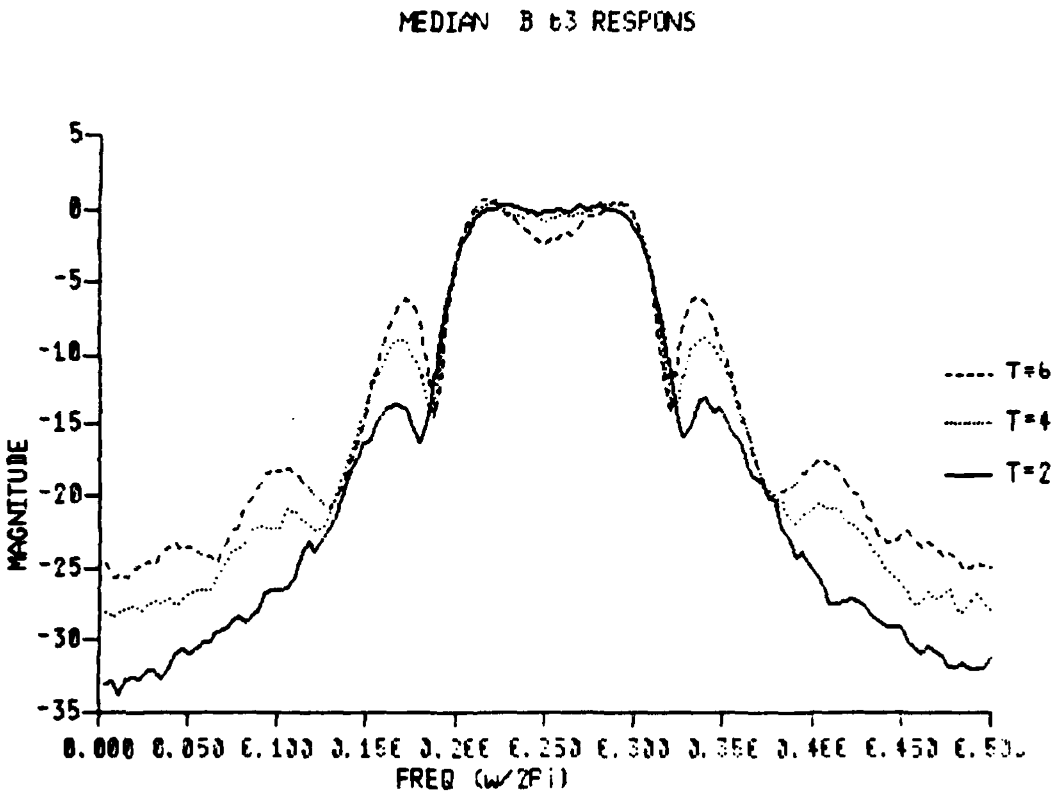


Figure 4 : α-TL BPF2500s L63 Transfer Functions

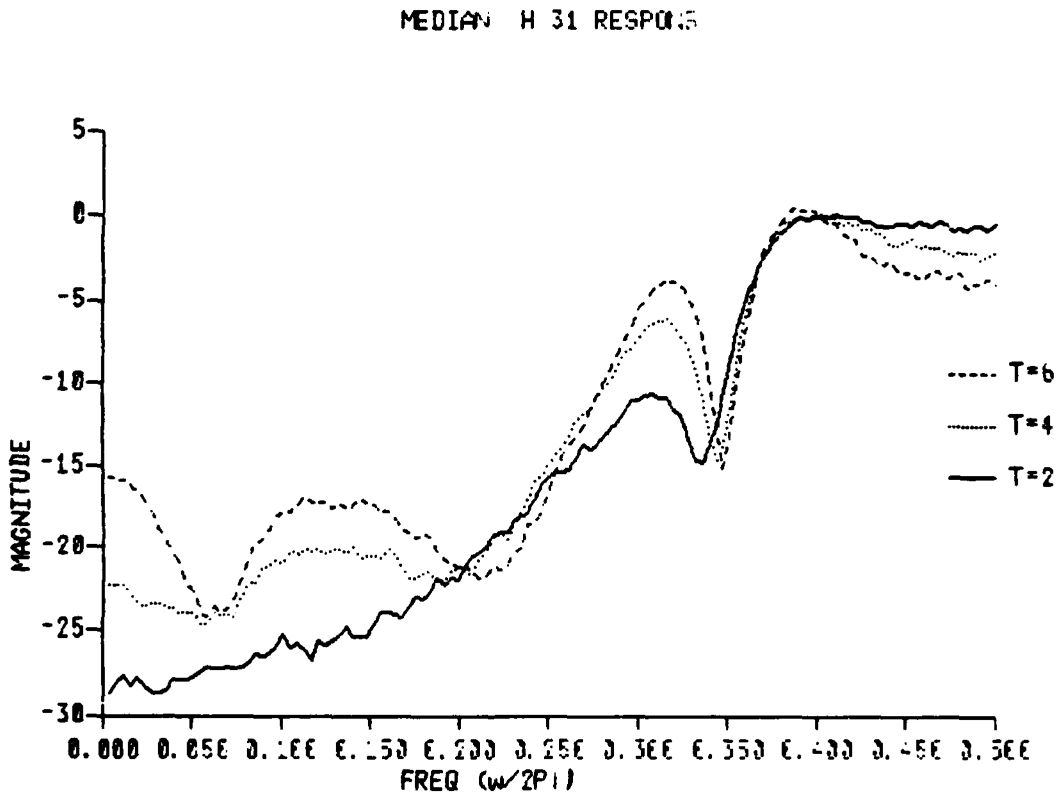


Figure 5 : α-TL HPF3500 L31 Transfer Functions

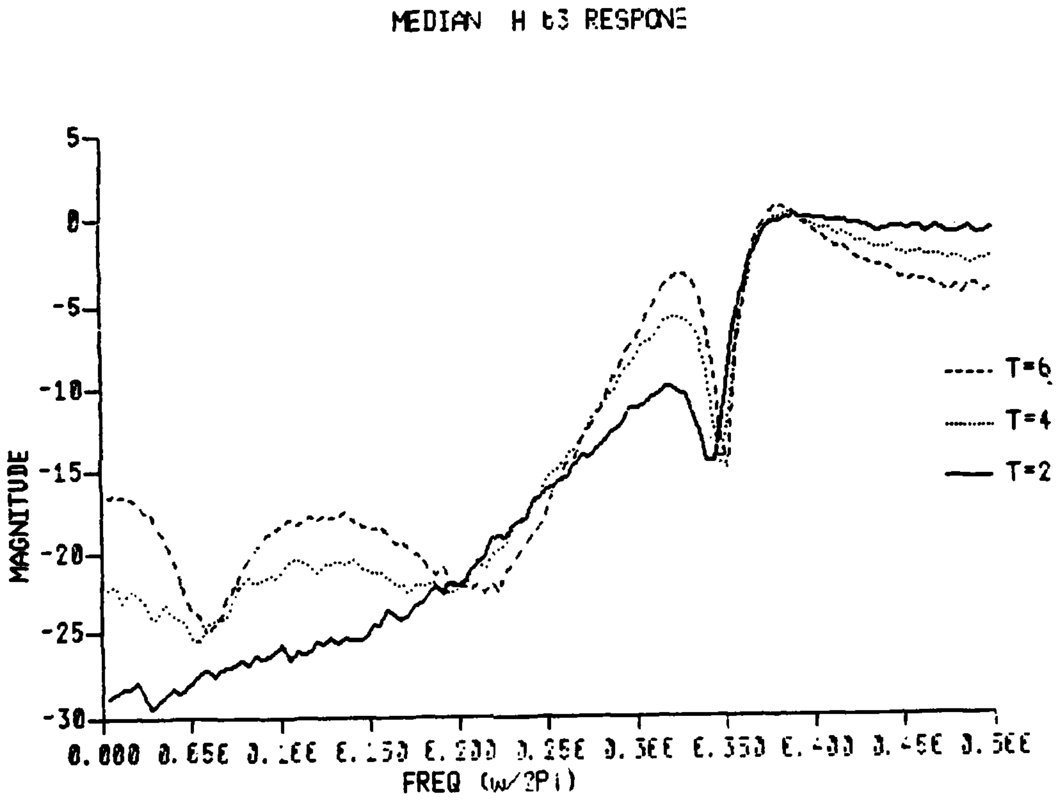


Figure 6 : α-TL HPF3500 L63 Transfer Functions

In order to complete the comparisons, note the maximum attenuations for each of the filters. For the BPF the maximum attenuation of the trimmed filters near 0 and .5 normalized frequency is about −32 to −33dB. This is for both the 31 element filter and the 63 element filter. Now compare this to the attenuation obtained by the actual linear filters. Figure 7 shows that the 31 element linear BPF has an attenuation of −47dB while Figure 8 shows that the 63 element filter has an attenuation of −84dB, an additional 37dB of attenuation! Yet the trimmed filters have nearly identical attenuation. A similar case exists for the HPF example. The linear filters of length 31 and 63 have stop band attenuation figures of −56dB and −102dB, respectively. A difference of 46dB! Yet the maximum attenuation of either of these filters when trimmed is only about −28dB. This limiting effect was seen in all of the examples in the simulation.

This effect is probably most closely related to one of the effects discussed at the beginning of this section. In the α−trimming process, the data elements that most often get trimmed are those that are modified by the largest linear coefficients. These coefficients are responsible for a large portion of the filter's overall shape. When the length of the filter is increased, the additional elements are typically very small and have a very fine effect. They only become significant when combined with the larger coefficients of the filter. Taking out the larger coefficients probably nullifies most of the subtle effect the smaller coefficients have in lowering the stop band.

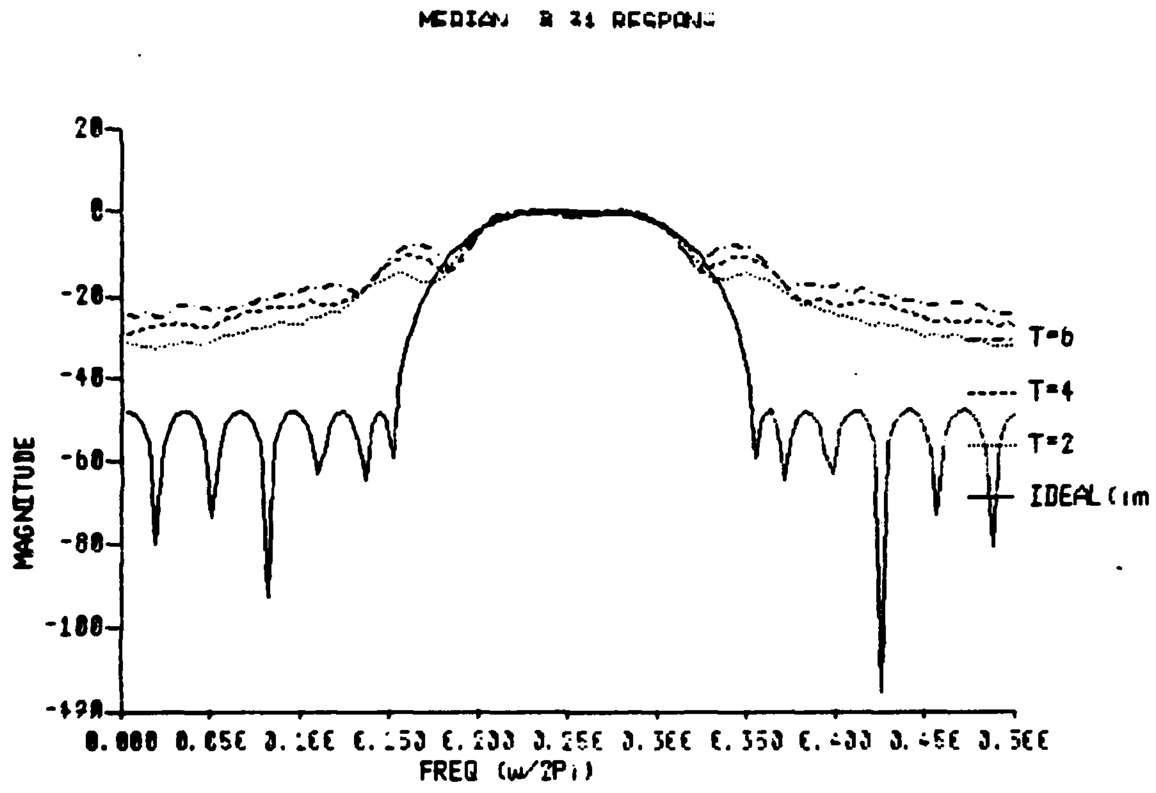


Figure 7 : α–TL BPF2500s L31 Transfer Functions plus Ideal Frequency Response

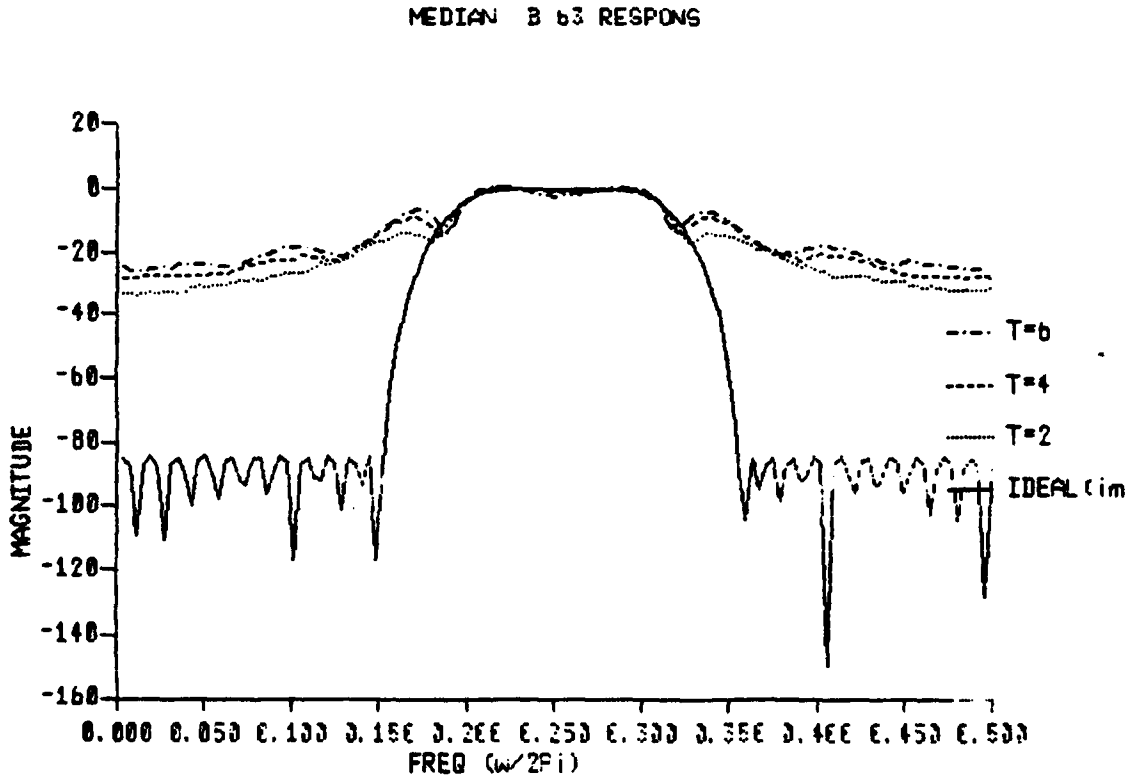


Figure 8 : α–TL BPF2500s L63 Transfer Functions plus Ideal Frequency Response

EFFECTS OF DESIGNED ROLLOFF

If you were watching closely in the previous section you may have noticed something strange that was not emphasized. In the filter examples given the linear band pass filters had attenuations of −47dB and −84dB for 31 and 63 elements, respectively. Likewise the high pass filters had corresponding attenuations of −56dB and −102dB in the stop band. Yet, when 2 elements were trimmed from the BPF the attenuation only dropped to ≈−33dB, while the same amount of trimming off the HPF caused the attenuation to rise all the way to ≈−28dB. The BPF suffered less degradation than the HPF! This seems counter-intuitive.

Taking a closer look by comparing the two 31 element filters we see that the attenuation of the BPF dropped from −47dB to roughly −33dB when two elements were trimmed. While the HPF, whose better original attenuation of −56dB, rose to about −28dB when the same trimming was done. Obviously there is some effect other than filter length involved here.

It turns out that the answer is in the coefficients themselves, or at least in their design. The main difference between the high pass and the band pass filter of the previous section is the width of the transition band. The BPF was designed with a steeper rolloff than the HPF. It may seem improbable that this is indeed the reason for the discrepancy in the stop band attenuation since the filters pass different frequency bands, so the following example is offered.

A second band pass filter was designed with the same, more gentle rolloff as the HPF recently discussed. The frequency response of the 31 element filter is shown in Figure 9. Note that the linear filter has a −56dB attenuation in the stop band, the same as the HPF of the previous section. In fact, in other filter designs it did not matter to where the actual pass band was moved, if the rolloff was kept at a width of 0.1 (normalized frequency) and the pass band was kept reasonably wide, then the attenuation in the stop band was very nearly −56dB for all examples. Now note in Figure 10 that the attenuation in the stop band does not go below about −22dB when 2 elements are trimmed. Yet this filter passes nearly the same frequency band as the previously discussed BPF whose attenuation went from −47dB to −33dB when 2 elements were trimmed.

It appears then that the coefficients for an FIR linear filter are more resistant to α−trimming when the filter is designed with a steeper rolloff, or at least the effect of α−trimming is dependent on the characteristics of the coefficients in some way. Somehow in these two examples the coefficients of the filter with the steep rolloff were more resistant to α−trimming than the other two shallower filters. (We can say this for α−trimming in general because, although it is hard to determine what is a good measure of the attenuation in the stop band for trimming greater than two elements, it is visually obvious that the attenuation for T=4 and T=6 was better for the steep rolloff BPF than for either of the other two filters to which it was compared.)

However, the previous two examples both used the same

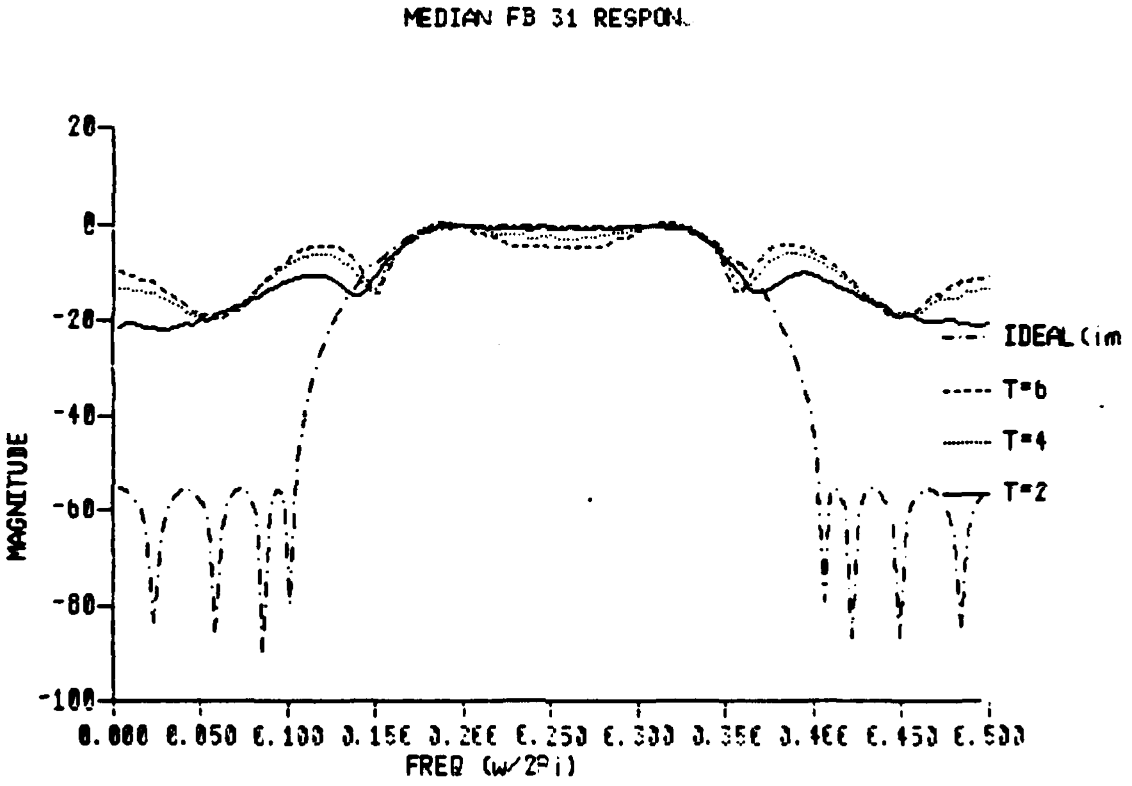


Figure 9 : α–TL BPF2500 L31 Transfer Functions plus Ideal Frequency Response

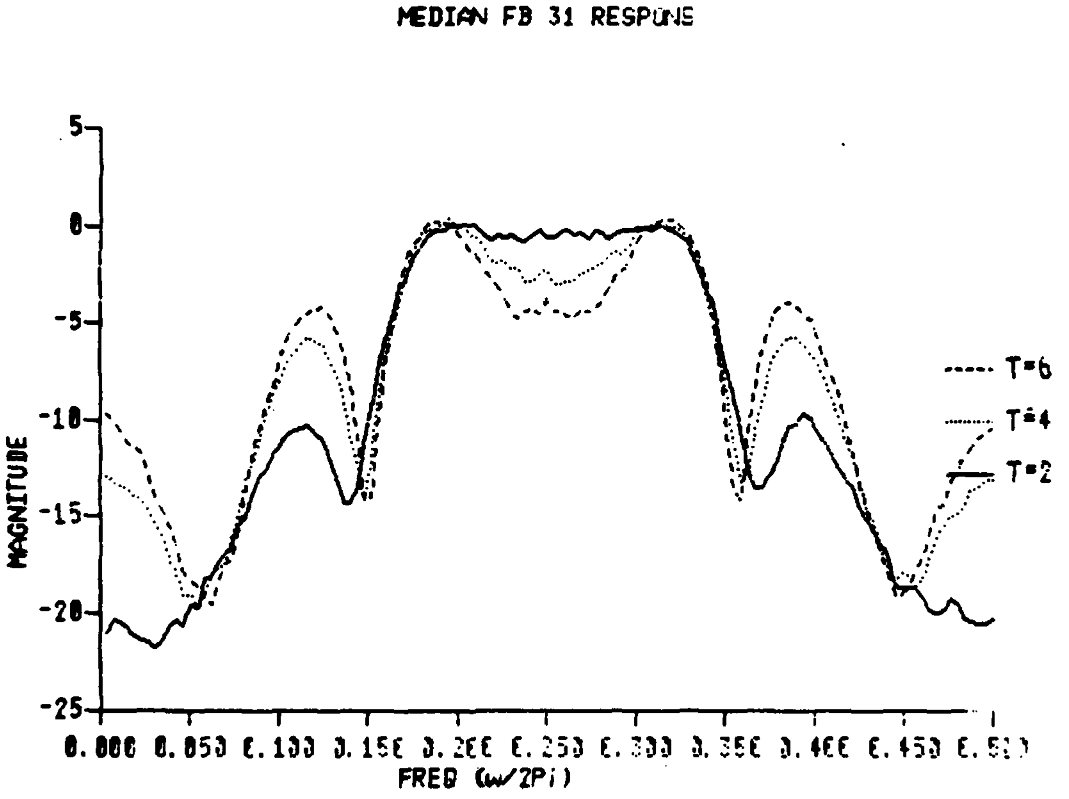


Figure 10 : α–TL BPF2500 L31 Transfer Functions

example of a steep rolloff filter for comparison. It could be just a fluke of the particular coefficients of that filter that the α–trimming happened to have less of an effect on it. So another steep rolloff filter was designed.

This time it was a LPF having a rolloff similar to the steep BPF, but a much narrower pass band. In this case the rolloff was 0.04 normalized frequency with a pass band of a mere 0.01. The designed attenuation in the stop band was −28dB for the 31 element filter and −52dB for the 63 element filter. The results for this filter were even better than for the steep BPF as can be seen in Figures 11 and 12. The deviation for the 31 element filter is only 1 or 2dB for each successive pair of elements trimmed and the deviation for the 63 element filter reaches the same levels only taking a part of the stop band to reach that point.

SYMMETRY PROPERTIES

A somewhat unexpected property of the α–trimming process was the high degree of symmetry it maintained. This symmetry came in two important forms. First, the graphs of each of the band pass filters in the previous section showed that the transfer functions maintained the original band pass filter's symmetry about the pass band. The attenuation was symmetric in each of the two stop bands.

This situation was not necessarily expected. It's existence shows that the α–trimming process does not have any frequency selectivity. The trimming in the α–TL filter does not modify the set of linear filter coefficients as does the α–trimmed mean filter.

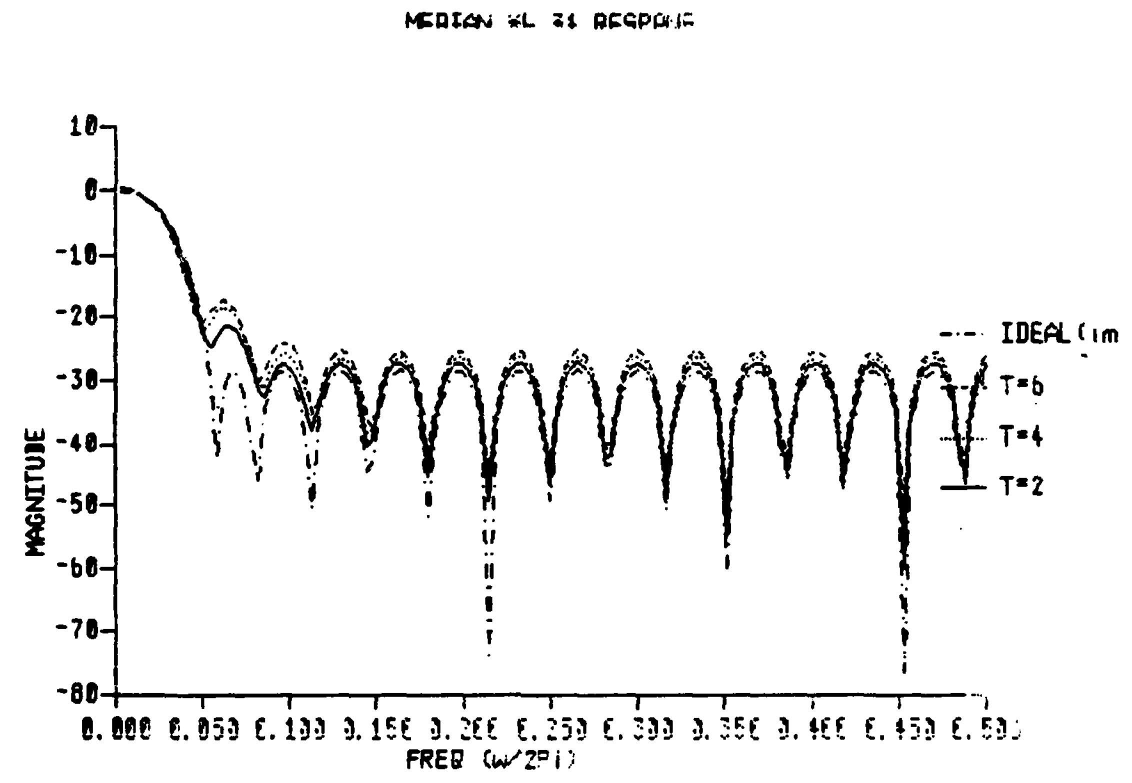


Figure 11 : α–TL LPF300 L31 Transfer Functions plus Ideal Frequency Response

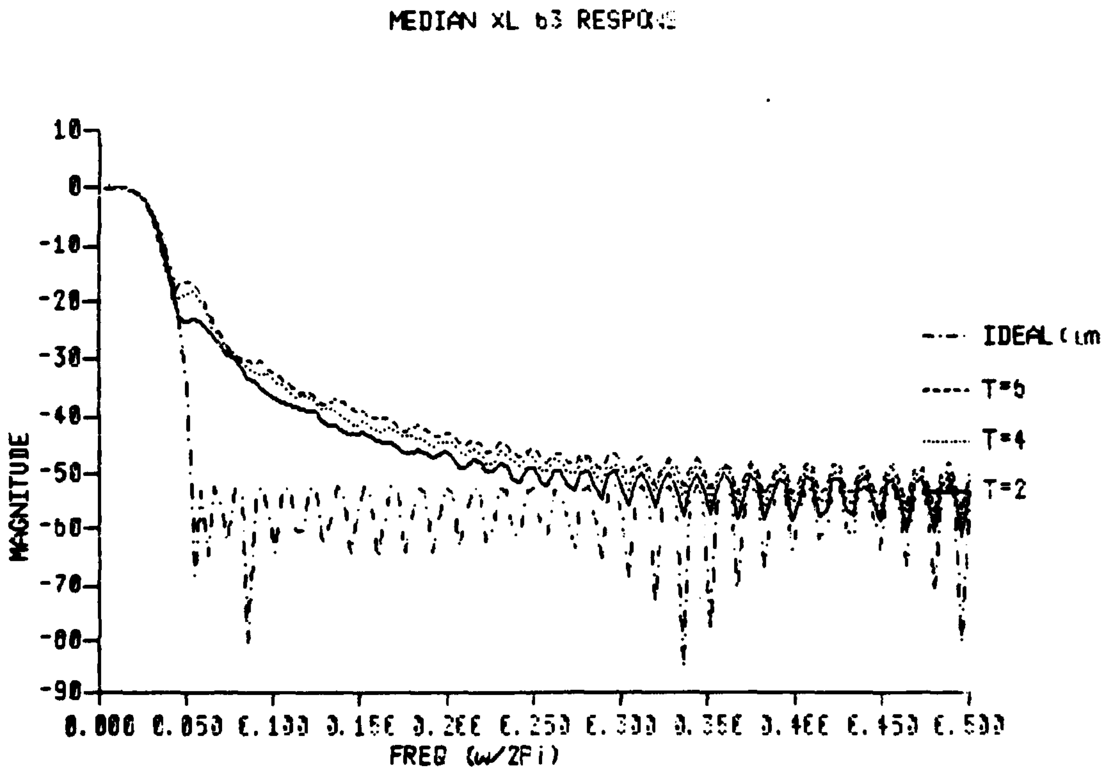


Figure 12 : α–TL LPF300 L63 Transfer Functions plus Ideal Frequency Response

The α-TM filter's coefficients are modified depending on how many elements are trimmed from the filter. No such modification occurs in the α-TL filter. Thus, it could easily be expected that α-trimming a linear filter may affect its frequency properties. The simulation shows that this is not the case, however.

Second, the other important symmetry property maintained was one dealing with similar filters. The LPF and HPF transfer functions shown in Figures 13 and 14, respectively, show how the α-trimming process maintains symmetry across the frequency spectrum. As one can see in Table 1, the coefficients of each of the filters have the same magnitudes. They only differ in sign. One might expect that the the filters would be trimmed differently, since the largest positive and negative elements are trimmed in the α-trimming process, causing different transfer functions.

Table 1: LPF1500 L9 and HPF1500 L9 Coefficients

LPF1500 L9		HPF3500 L9
-0.04739	= h(1) = h(9) =	-0.04739
0.05110	= h(2) = h(8) =	-0.05110
0.15307	= h(3) = h(7) =	0.15307
0.25926	= h(4) = h(6) =	-0.25926
0.30437	= h(5) =	0.30437

The explanation for this symmetry seems fairly straightforward. The noise used to calculate the transfer functions

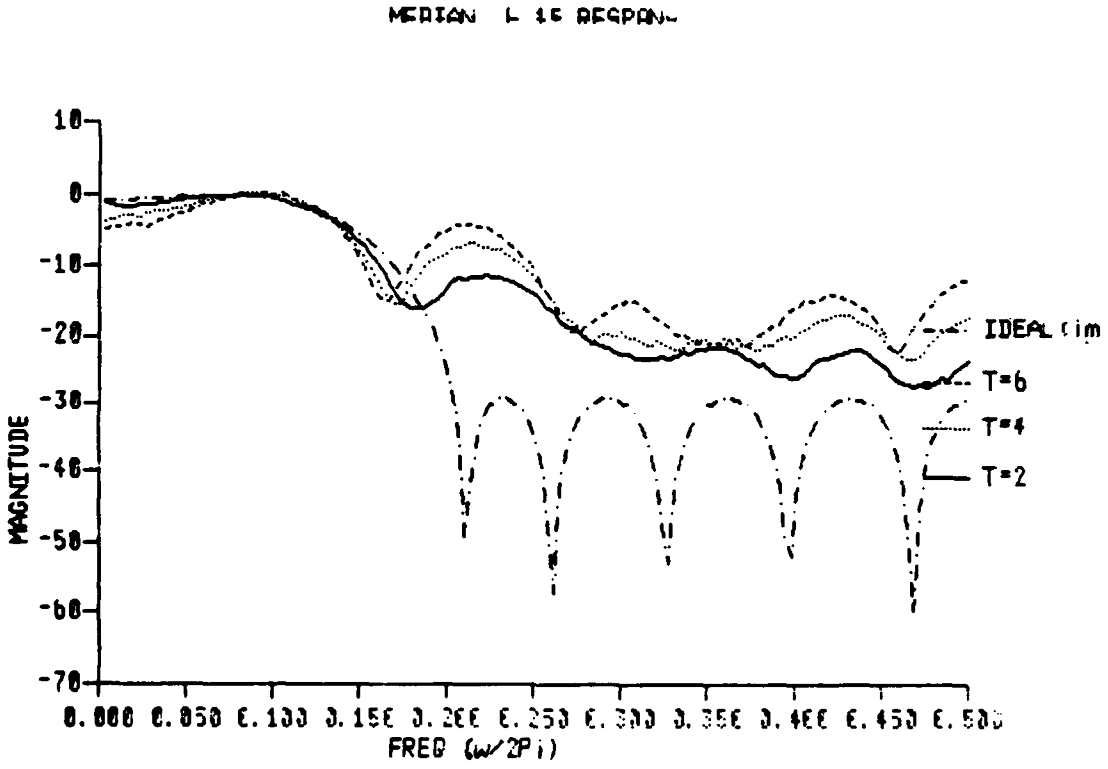


Figure 13 : α–TL LPF1500 L15 Transfer Functions plus Ideal Frequency Response

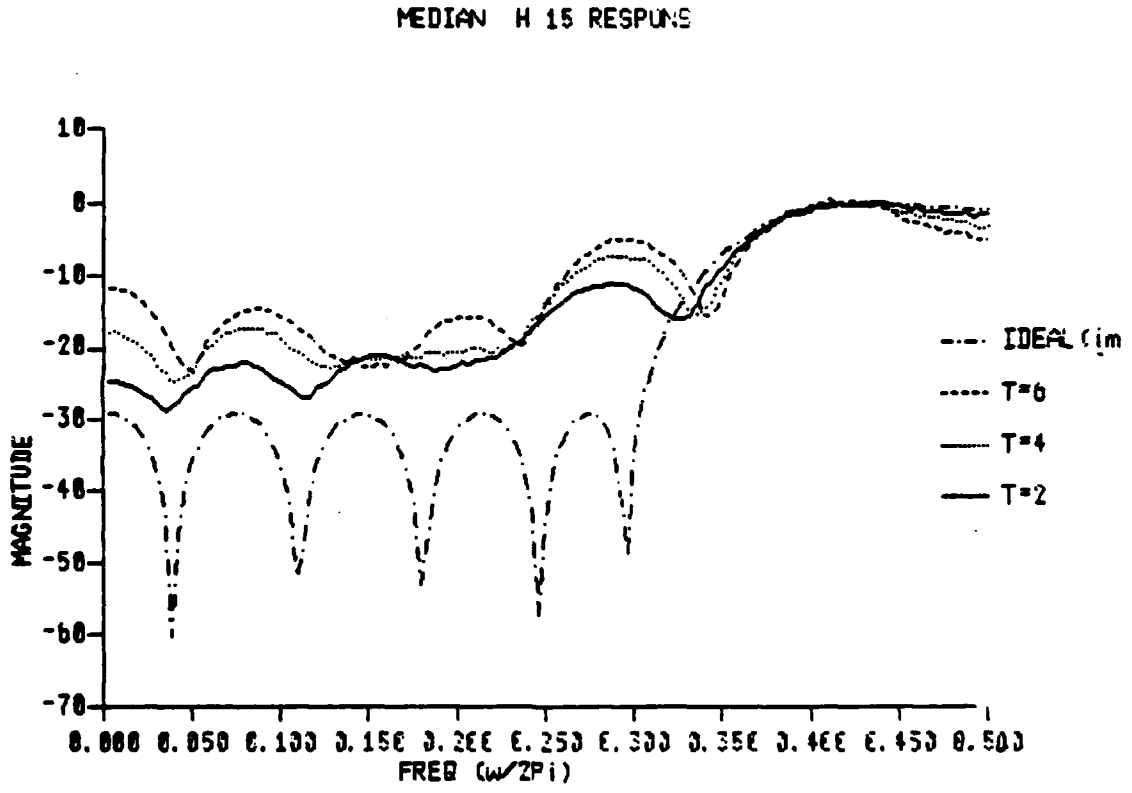


Figure 14 : α−TL HPF3500 L15 Transfer Functions plus Ideal Frequency Response

is zero mean and Gaussian and is, therefore, equally likely to be negative as positive. Thus, on the average, the effects of a difference in signs of the coefficients will not be a factor in the long run and only be a factor in determining the frequencies passed in any given interval, high frequencies or low frequencies depending on the filter. An interesting test would be to give the noise a dc bias so that every data point was the same sign. The trimming symmetry that these two filters exhibit may suffer under these input conditions.

FILTER GAIN

A troubling problem with the α-TL filter is the uncontrolled gain of the system. The gain calculation outlined in Section III does a good job of estimating the gain of an α-TL filter only if the linear filter is a low pass filter. Experimentally, the gain factor correction was very good for the LPF case. However, it was totally unpredictable for any filter not passing dc. This gain problem showed up in the transfer functions as a spike at 0 Hz. (This spike was not evident when the median was used to select the transfer function, only the mean. The distribution of values at the dc point has a much larger deviation than at any other frequency for all of the filters in the simulation. See Appendix A for more details.)

Interestingly enough, though, an unsuspected consistency was found while attempting to calculate the error to evaluate the performance of an α-trimmed BPF in rejecting outliers. It was necessary too correct the gain problem in order to calculate the

MSE between an ideal signal and the noise corrupted signal filtered by the α–TL filter. So the output was modified by a minimum MSE calculation. This minimization led to the calculation of the required normalization factor. Surprisingly, the normalization factor (same as the gain factor g) for a given α–TL filter remained fairly constant regardless of the noise on the input signal. The normalization factors for two of the band pass filters are given in Figures 15 and 16. The values in these figures were obtained by filtering the ideal signal—a sinusoid in the pass band of the filter—with the α–TL BPF and then calculating the required normalization factor to minimize the MSE between the filter output and ideal input.

The normalization factors calculated when there was noise on the input signals were consistently smaller than the values in the

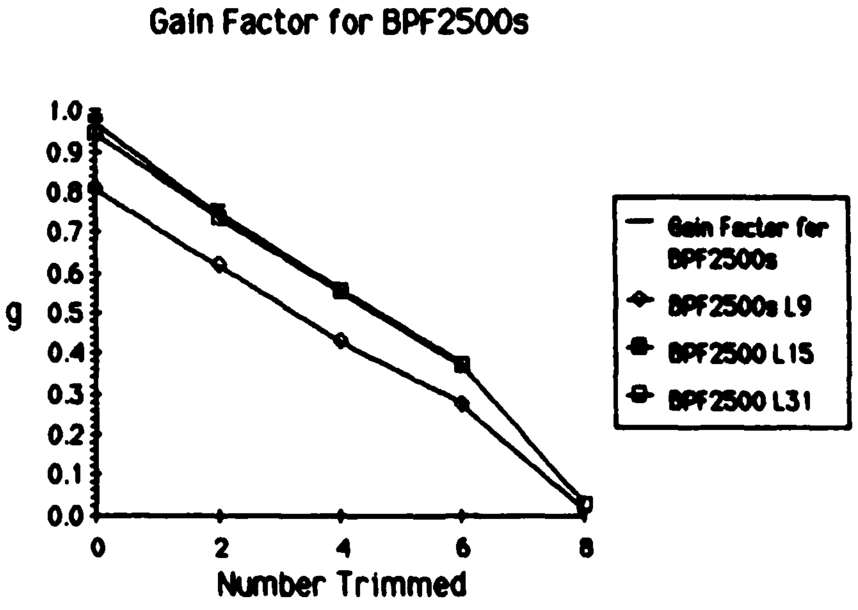


Figure 15 : Gain Factor for α–TL BPF2500s

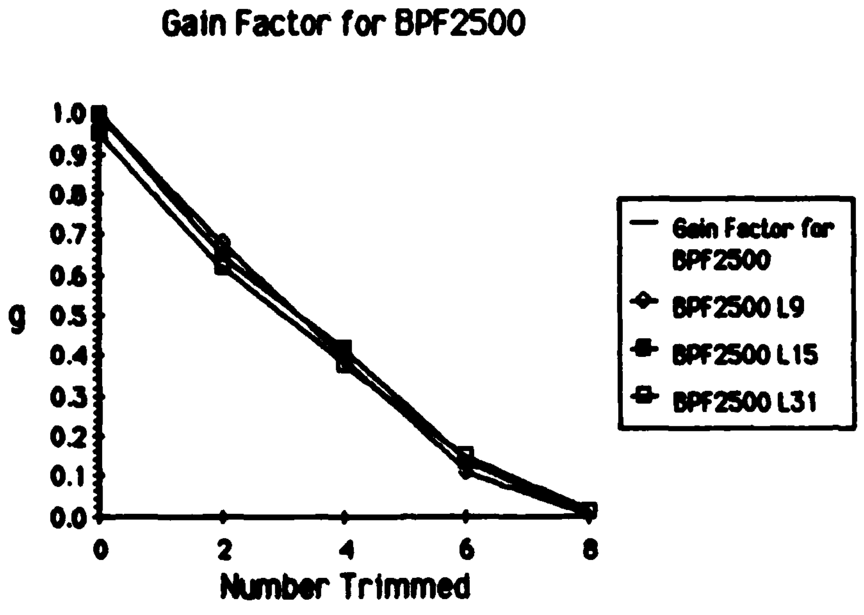


Figure 16 : Gain Factor for α-TL BPF2500

figures, but right in the same neighborhood. This would be expected since the noise on the signal would almost certainly add slightly to the MSE between the ideal and the corrupted signal.

OVERALL FREQUENCY PERFORMANCE

One of the items that has not been stressed very much is the actual measurement of the frequency performance in the stop band of the α-TL filters. Where a reference has been made to a value for attenuation in the stop band of an α-TL filter, the measurement was made at the edge of the filter response where the attenuation had reached its best point. An important measurement, though, is how good the attenuation is at the first sidelobe. For the original set of linear FIR filters these two measurements were the same since the attenuation was uniform across the stop band.

This is far from the case when measuring the attenuation in the first sidelobe of an α-TL filter. As can be seen in Figure 17 this performance measure leads to some very poor results.

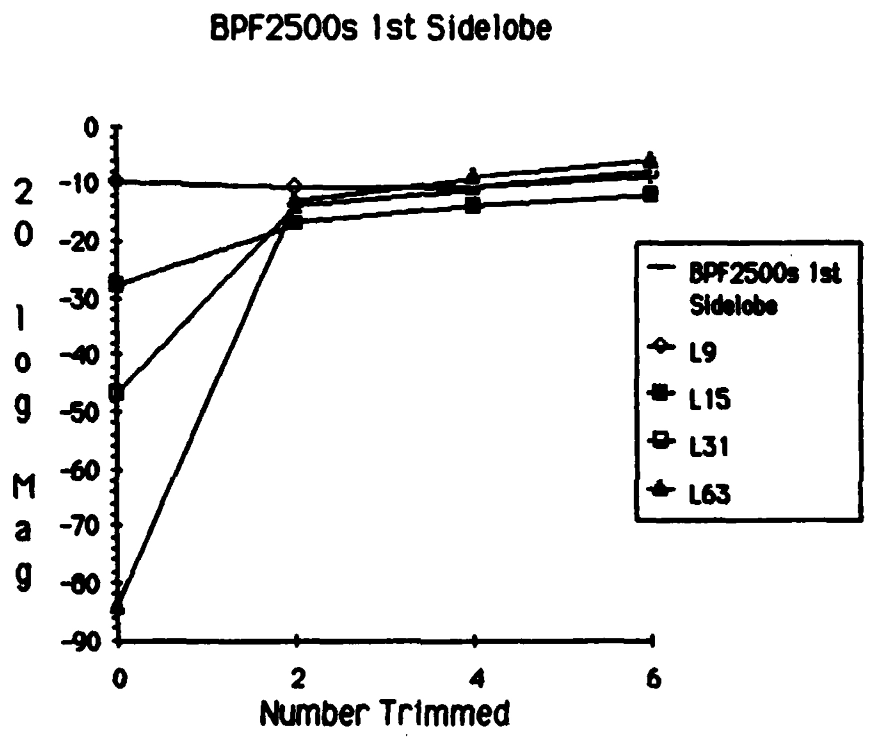


Figure 17 : First Sidelobe Level of BPF2500s

This figure shows that the most serious loss of attenuation occurs for the first two elements trimmed. After that the loss appears to be fairly linear and not nearly so severe. The first sidelobe is most affected in the long filter lengths. Since the sidelobes become more numerous and skinnier, the loss of attenuation shows up more clearly for the very narrow first sidelobes of the long filters.

This makes it difficult to say exactly how bad the degradation of frequency performance is for the α–TL filter. The attenuation is much better than the level at this first sidelobe for the majority of the stop band. Yet a significant amount of energy is passed through the filter in this first sidelobe that would be attenuated in the corresponding linear filter. About all that can be said for sure is that the frequency response of the linear filter suffers quite a bit when the filter is α–trimmed. The longer the linear filter and better the stop band attenuation, the more α–trimming degrades the performance.

B. OUTLIER RESISTANCE

The property of outlier resistance was one of the major design considerations in all of the filter designs discussed in Section II. One would therefore expect, for the low pass filter case embodied in all of those designs, that adequate noise smoothing with impulse rejection has been achieved with some degree of success. It seems unlikely that a new filter design which has outlier resistance as only half of the major filter objectives—the α–TL filter—would be able to compete with any of the previously mentioned filters. On the other hand, there is nothing to compare a band pass or high pass α–TL filter to except the corresponding linear filter. In this case one would hope that the α–TL filter could outperform the linear filter in rejecting impulses.

VISUAL EVALUATION

In visually inspecting the performance of the α–TL filter, however, it is not practical to look at a high pass or even band pass signal. No meaningful information is readily derived from trying to do so. In order to determine the effectiveness of the α–TL filter in rejecting impulses by a visual inspection it is necessary to limit the investigation to the LPF case. This approach also allows a comparison of the α–TL filter to several of the other filter designs discussed in Section II. Specifically, a comparison of the low pass α–TL filter will be made to the α–TM filter (and the mean and median filters, noting that these are special cases of the α–TM filter having 0 and N–1 elements trimmed, respectively).

The input signal that best showed the filters' performance was a signal similar to the one used by Lee and Kassam in one of their papers on order statistic filters.[1] The signal has several edges and several monotone regions. The signal is shown in Figure 18, before and after it is corrupted by white Gaussian noise, $\sigma^2 = 0.2$. Figure 19 shows the same signal corrupted by white Laplacian noise, $\sigma^2 = 0.5$. This signal is known as signal 1 (SIG1) and the two corrupted signals are known as SIG1G (Gaussian) and SIG1I (impulsive). A third version of this signal, SIG1IG, is SIG1I plus white Gaussian noise, $\sigma^2 = 0.1$, and is shown in Figure 20.

The filters used in the simulation for the visual evaluation were LPF300 L9 and LPF1500 L9. The 9 element filters were chosen because they were short compared to the features of the signal. The LPF300 signal is particularly interesting because it was

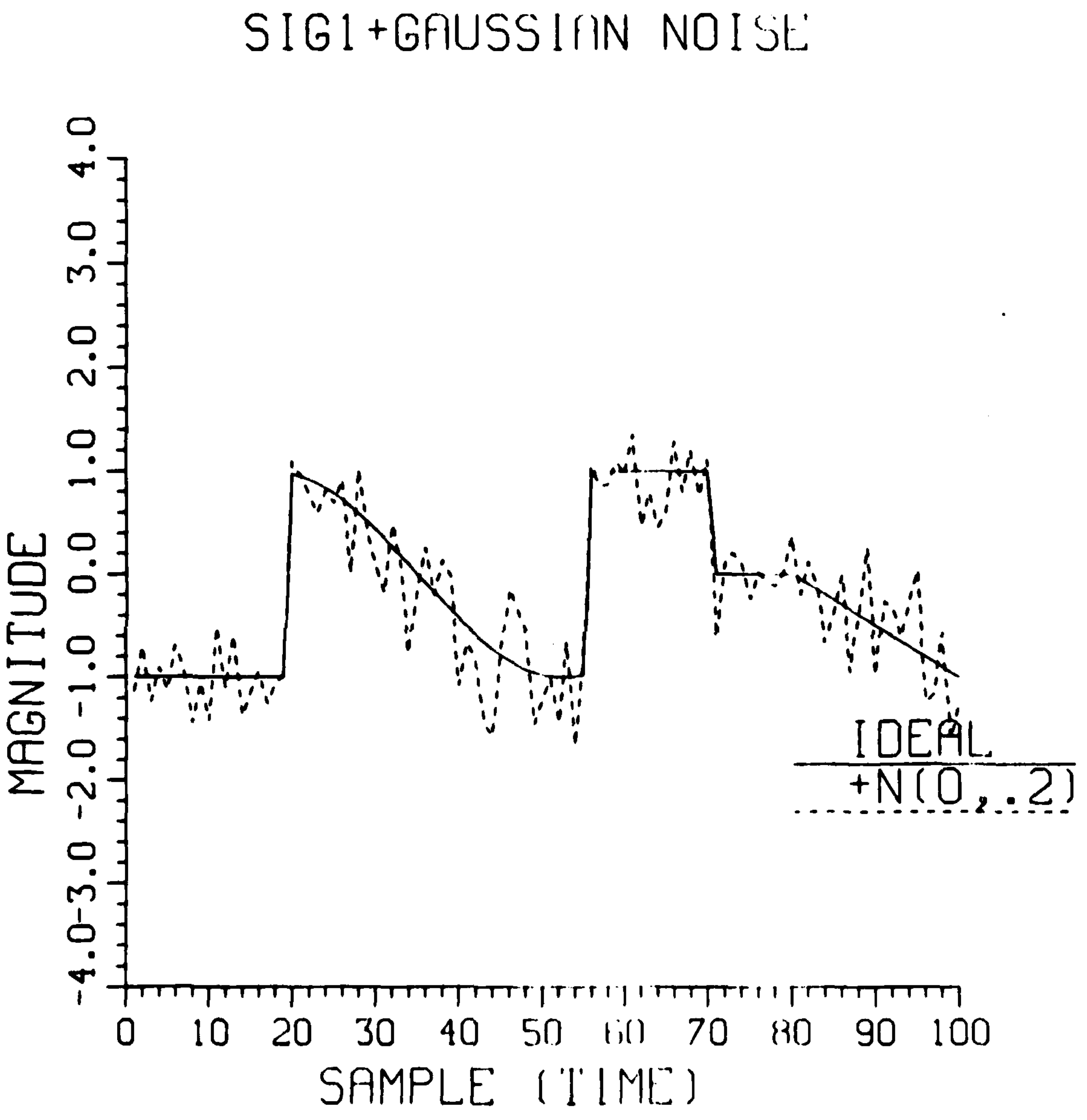


Figure 18 : SIG1G - signal 1 + noise ~N(0,0.2)

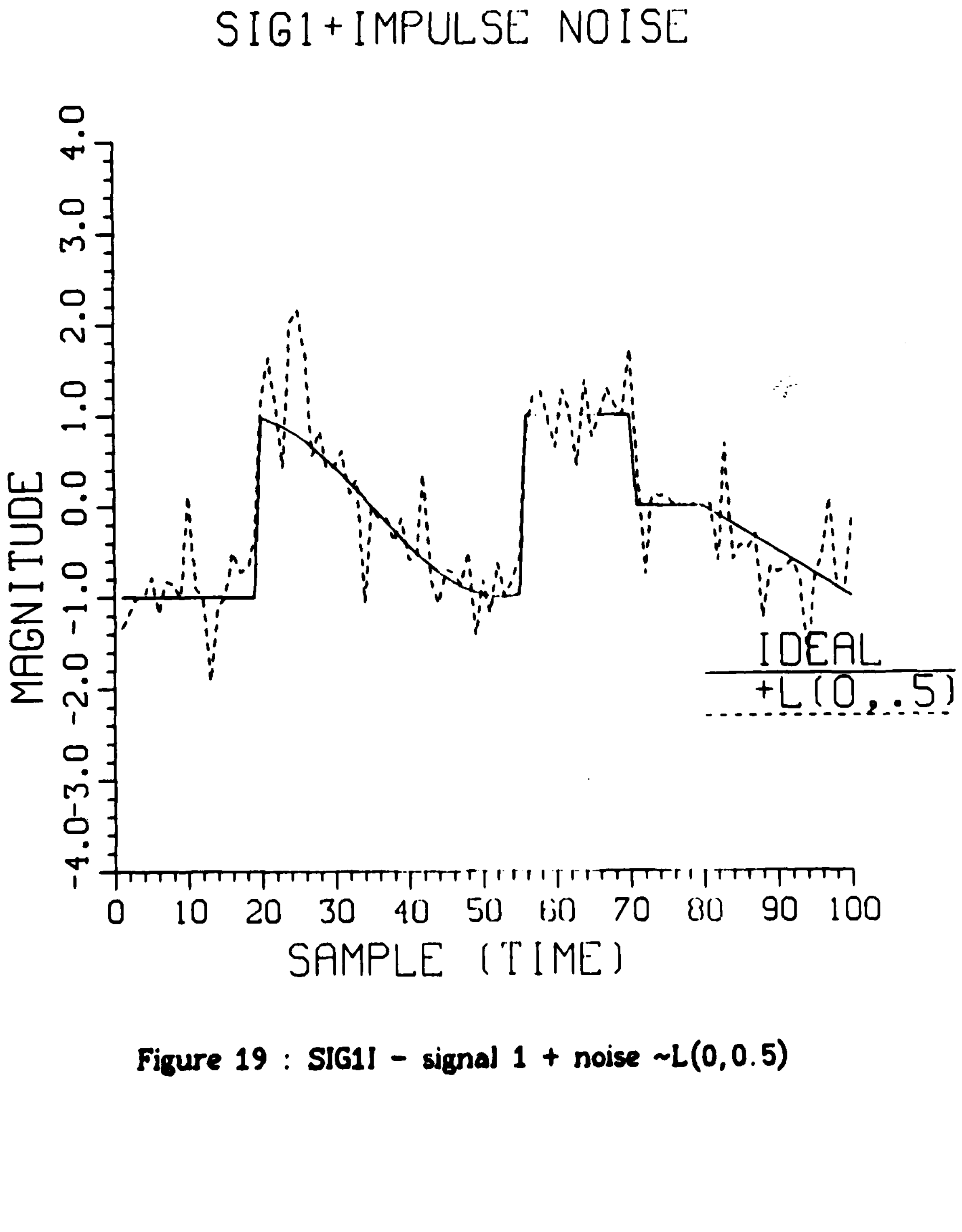


Figure 19 : SIG1I - signal 1 + noise ~L(0,0.5)

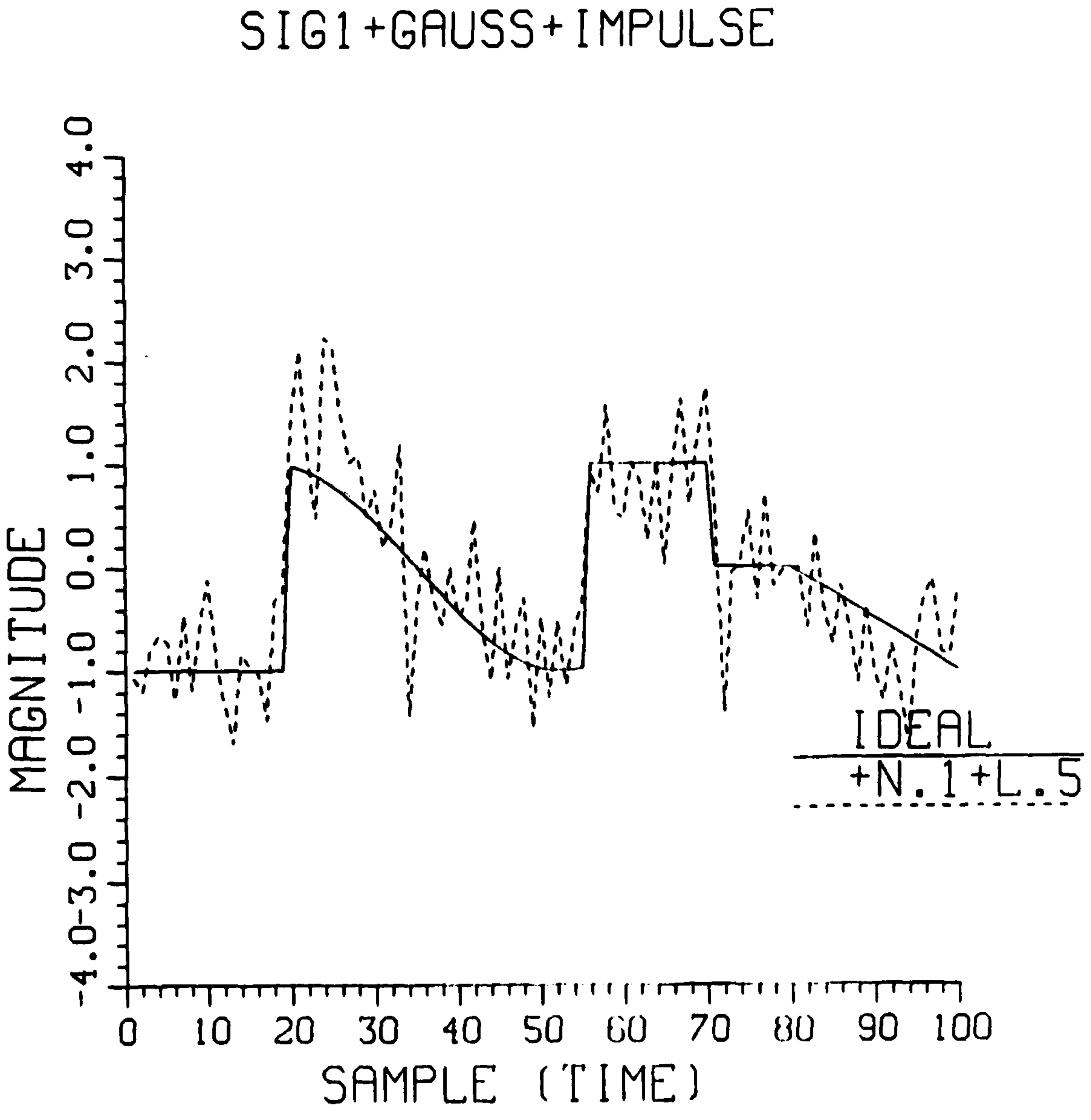


Figure 20 : SIG1IG - SIG1I + noise ~N(0,0.1)

designed to have a transfer function very similar to that of a mean filter, having a similar pass band width and rolloff.

The results of SIG1G being filtered by a mean and median filter of length 9 and α–TM 9T6 are shown in Figures 21, 22 and 23, respectively. As expected, the α–TM is a very good compromise between the noise smoothing of the mean filter and the edge preservation of the median filter. Figure 24 shows that the LPF300 L9 filter, even though it has a similar transfer function to the mean filter, does not do as good a job smoothing the Gaussian noise, but still introduces the same smearing of edges. This is due to the nearly constant coefficients of the filter. However, if the LPF300 L9 filter is α–trimmed 6 elements, the results are very comparable to the α–TM filter of the same length and trimming. The α–TL filter fails to smooth the noise as well near the end of the sinusoidal section, but matches the flat pulse much better as shown in Figure 25. The LPF1500 L9 filter lets through too much of the noise, even when α–trimmed 6 elements as shown in Figures 26 and 27, respectively.

Of a more primary interest is how these filters perform in the presence of impulsive noise. The mean, α–TM and median filters perform as expected in Figures 28, 29 and 30, respectively. The important spots to watch in all of these figures are the doublet type impulses in the first flat region, the single and double width impulses at the top of the sinusoid, the pair of opposite pulses in the middle of the sinusoid region, the doublet type impulses at the falling edge of the step and the string of five impulses on the final

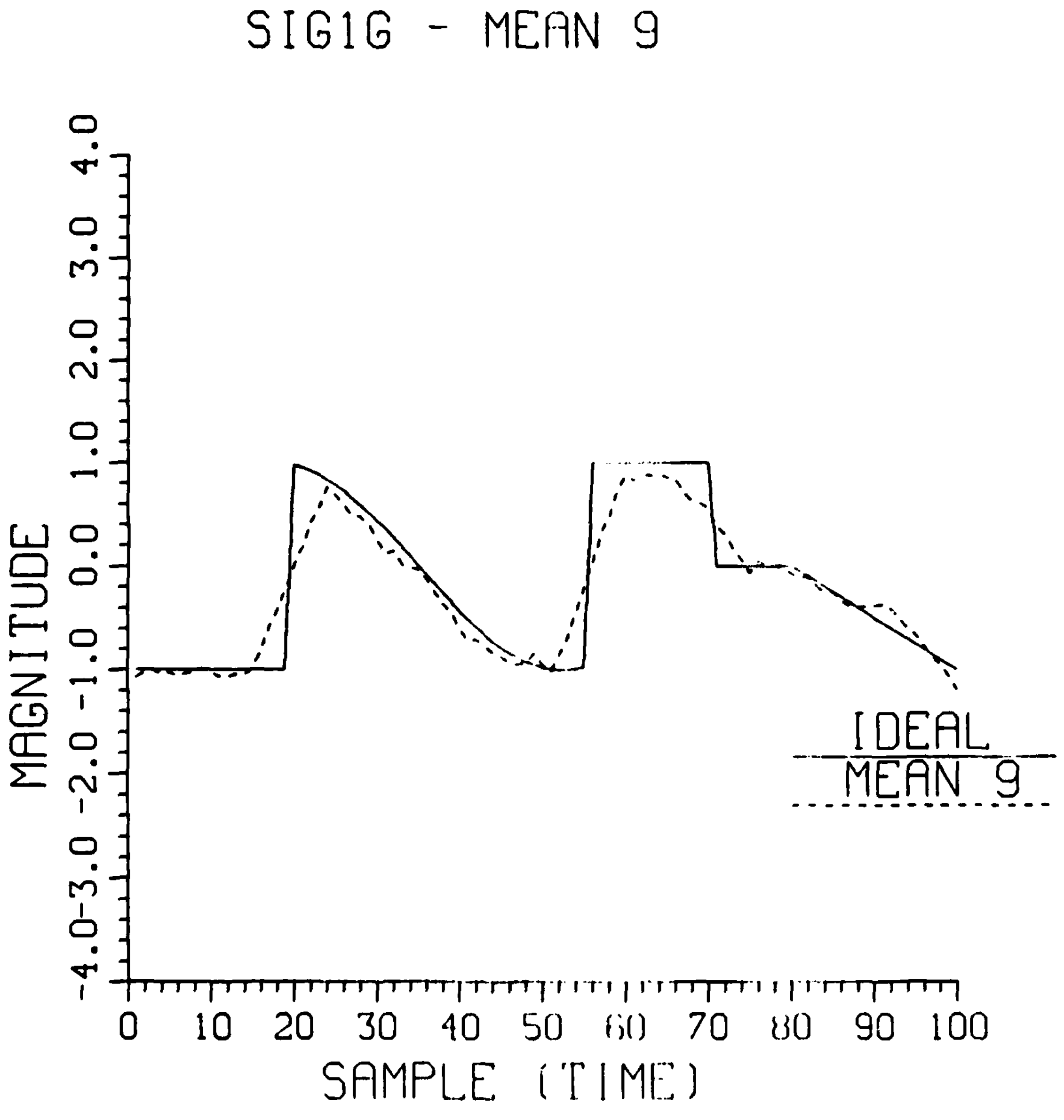


Figure 21 : SIG1G Filtered by Mean 9

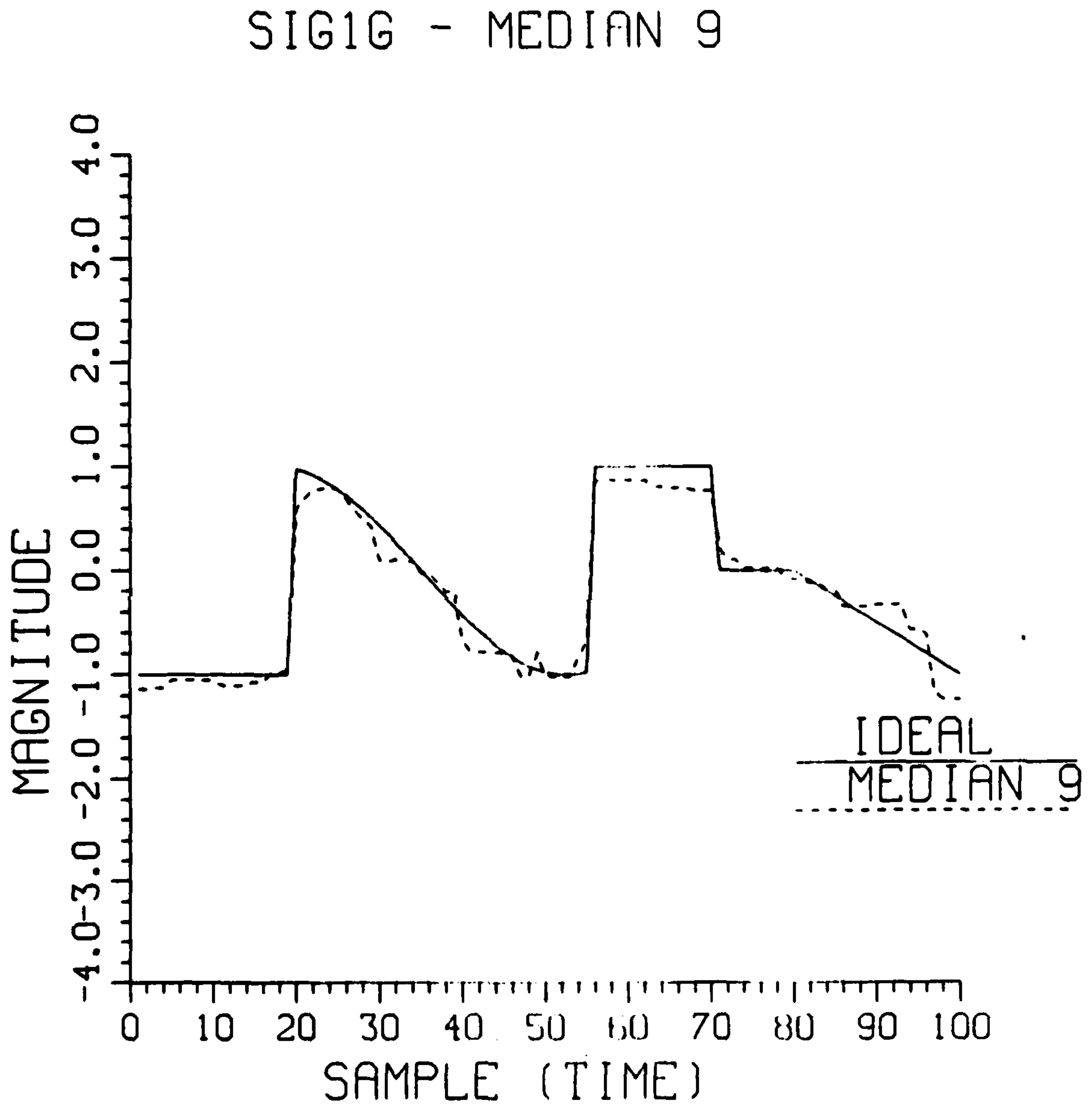


Figure 22 : SIG1G Filtered by Median 9

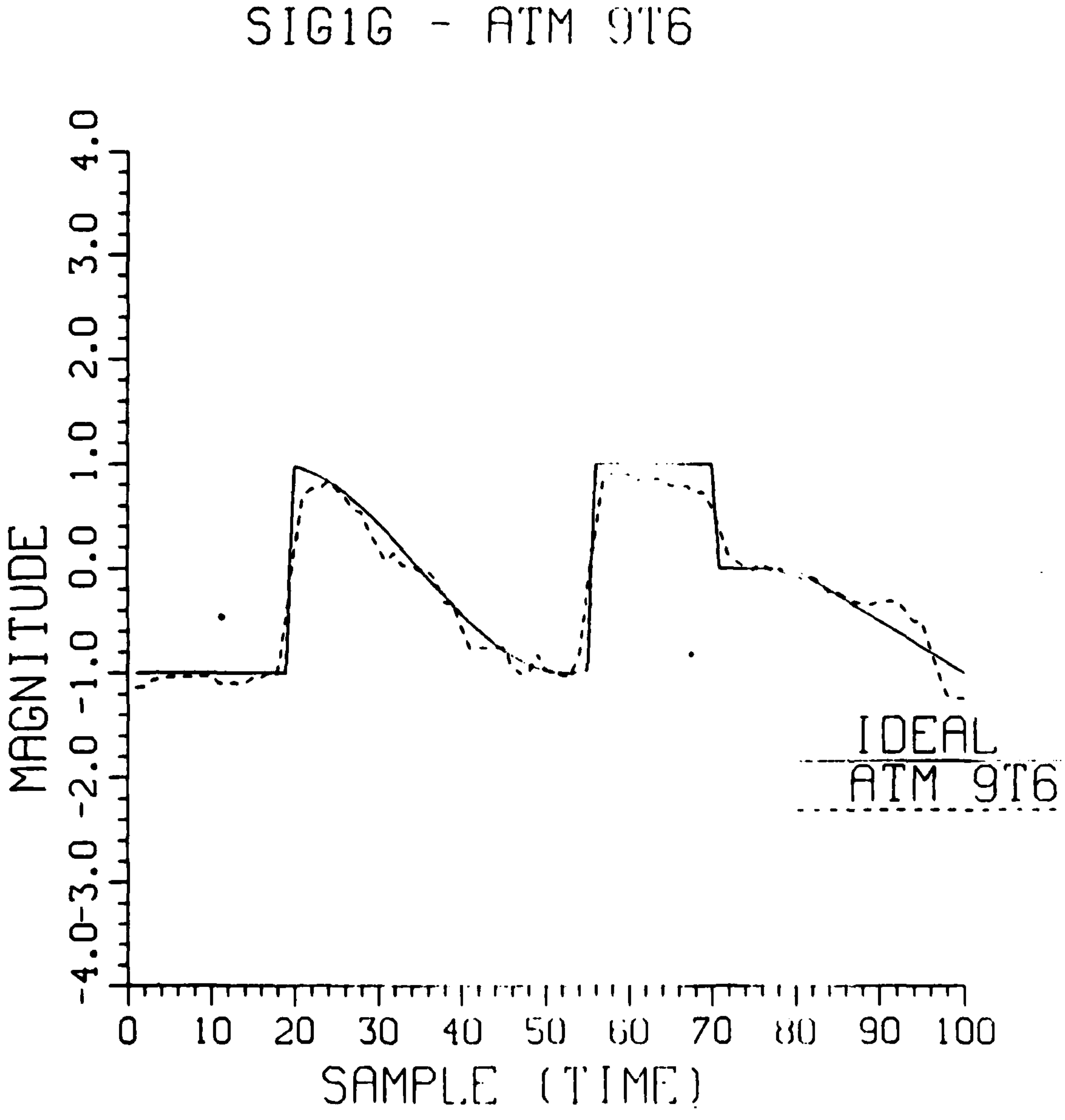


Figure 23 : SIG1G Filtered by α–TM 9T6

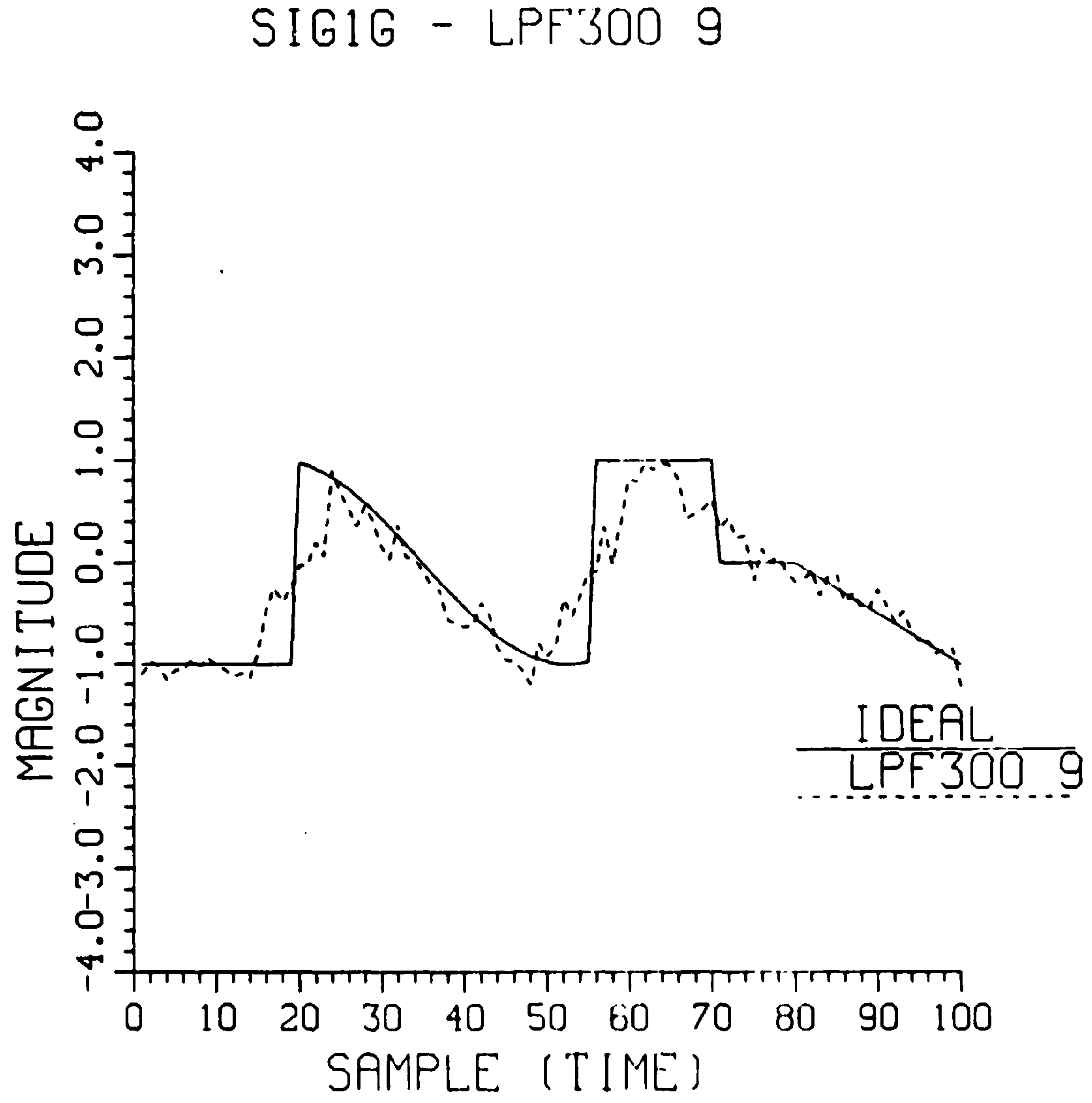


Figure 24 : SIG1G Filtered by LPF300 L9

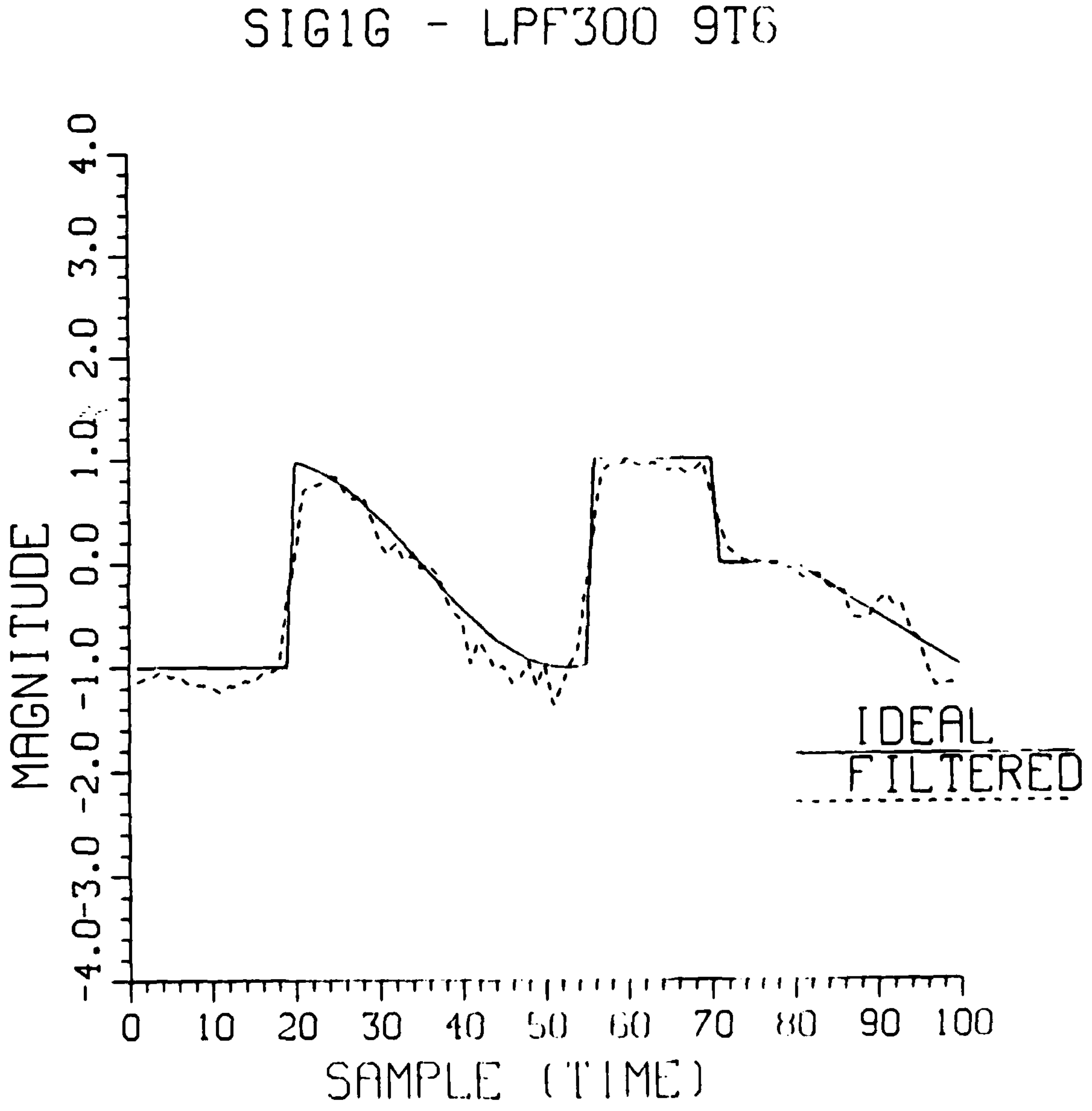


Figure 25 : SIG1G Filtered by α-TL LPF300 9T6

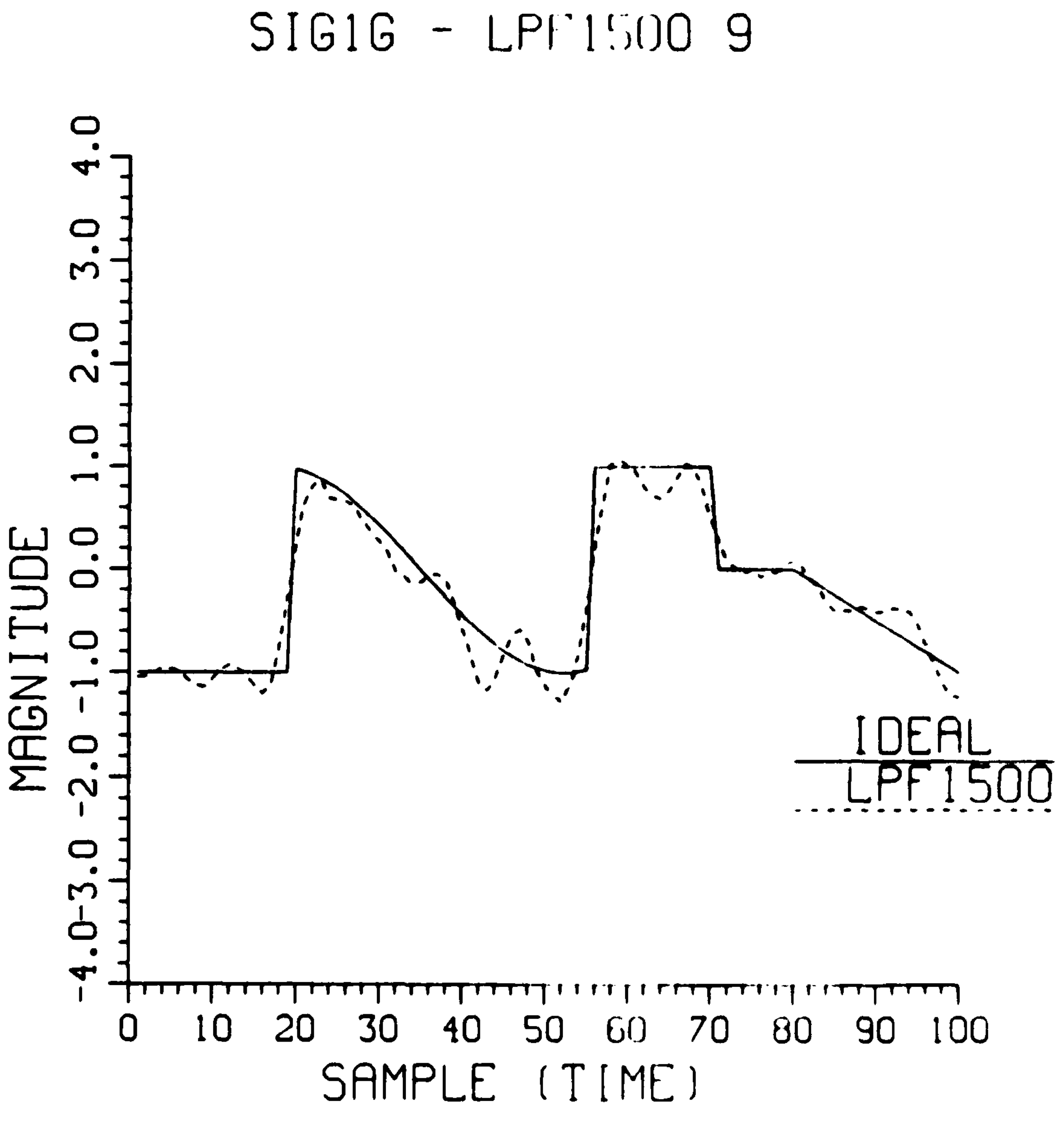


Figure 26 : SIG1G Filtered by LPF1500 L9

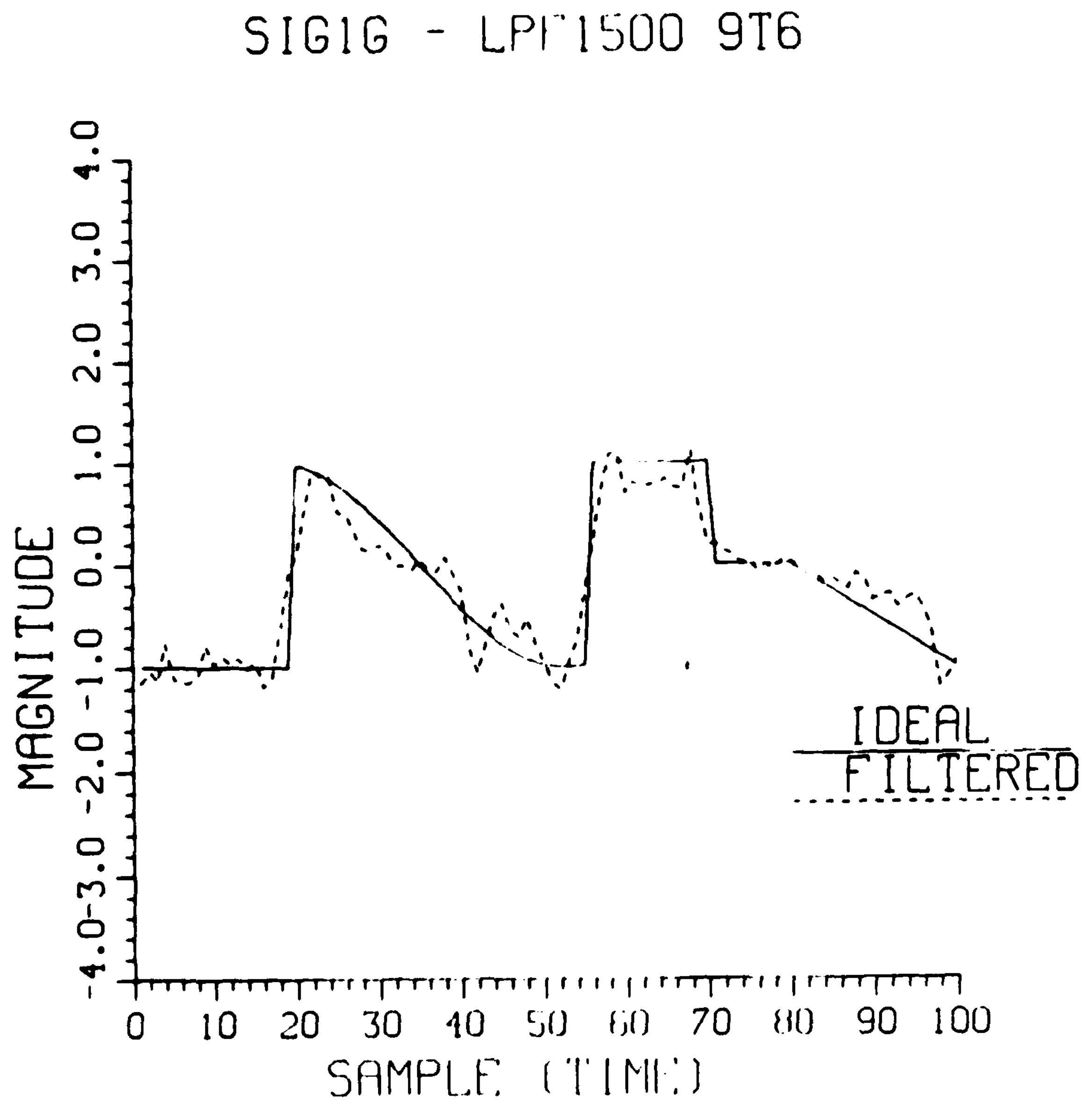


Figure 27 : SIG1G Filtered by α-TL LPF1500 9T6

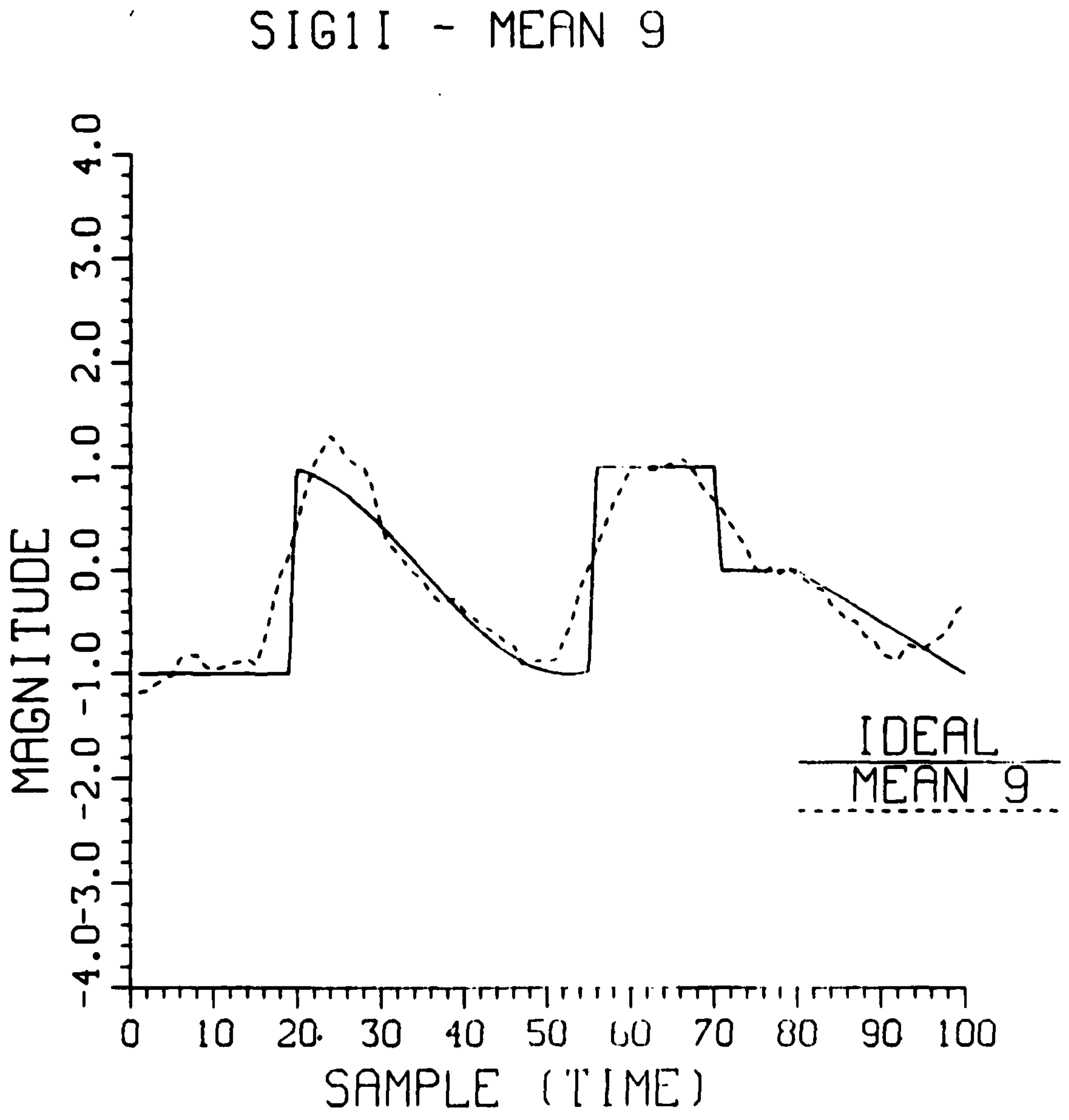


Figure 28 : SIG1I Filtered by Mean 9

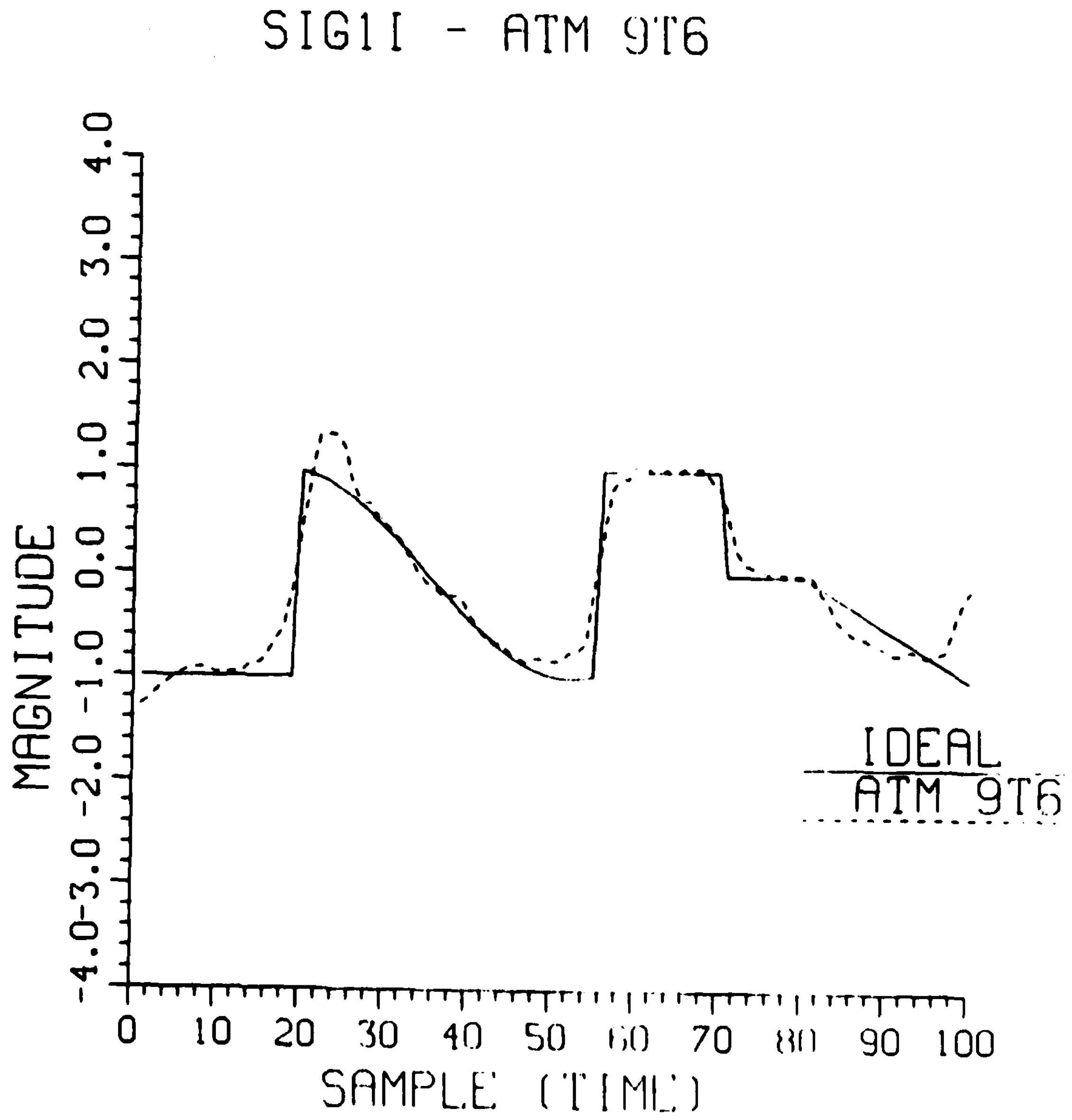


Figure 29 : SIG1I Filtered by α–TM 9T6

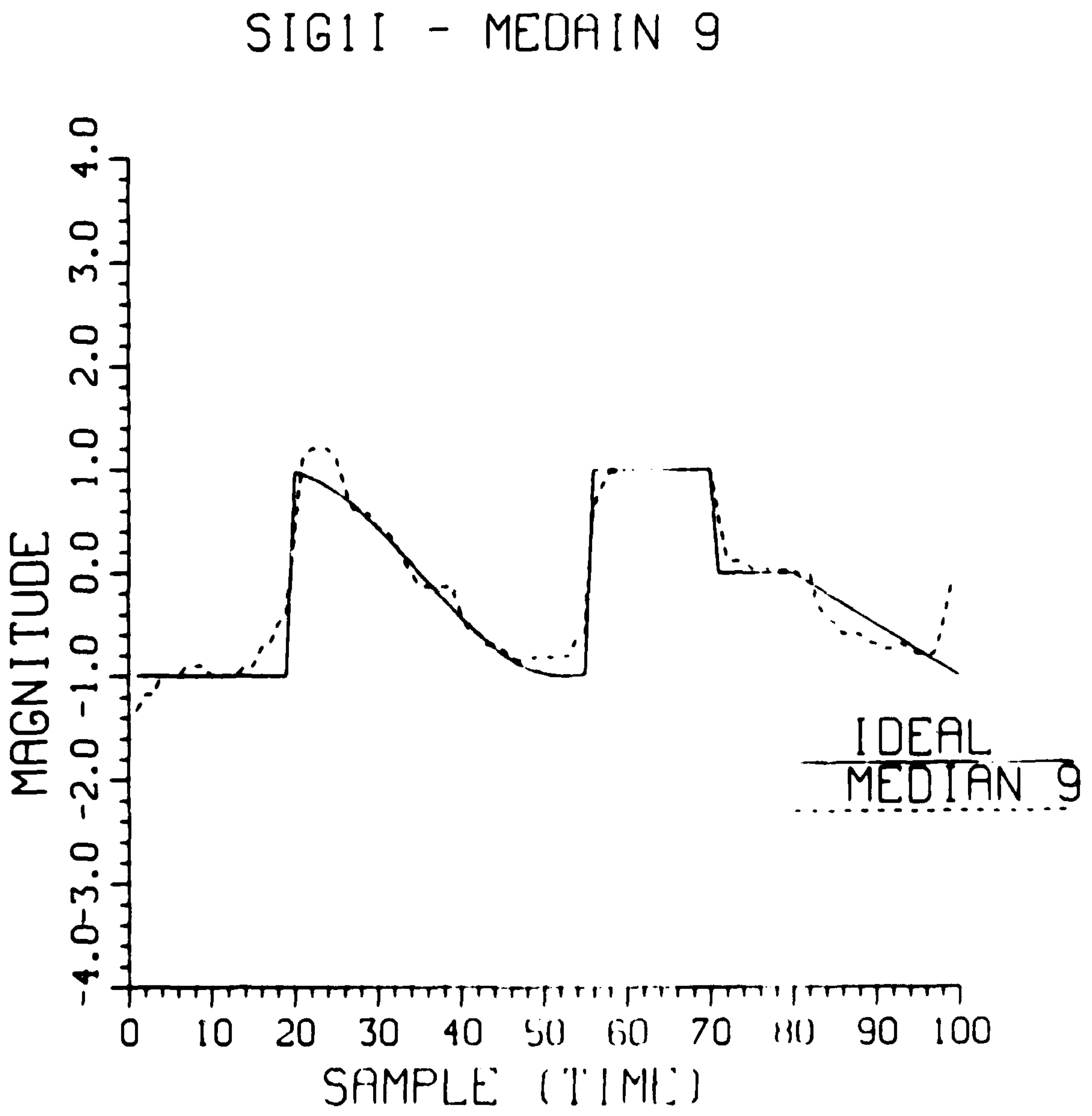


Figure 30 : SIG1I Filtered by Median 9

ramp. Note in each of the three preceding figures, the filters dealt with all of the impulses fairly well except the impulses at the top of the sinusoid and those on the final ramp.

Amazingly, the LPF300 L9 filter takes care of the effects of all of the impulses quite well as shown in Figure 31. However, this filter still lets through too much small level noise and introduces quite a bit of distortion to the signal. It can be seen in Figure 32 that when this filter is α–trimmed 6 elements the distortion and low level noise are taken care of, but the pair of impulses on the sinusoid have quite an effect. The real power of α–trimming is seen on the LPF1500 L9. In Figures 33 and 34 one can see that this linear filter still passes too much high frequency noise, but the distortion is smaller than that of LPF300 due to the larger pass band being better able to follow the edges in the signal. Also, trimming six elements has some degree of success against the outliers. In fact, this is the only filter that can be said to have taken out the majority of the effect of the pair of impulses on the top of the sinusoid.

Although these graphs give a feeling of what can be expected from an α–TL filter compared to one of the more traditional filters, it is still difficult to see what is actually happening in the α–TL filter. One example that delivered some really tangible visual results is shown in Figure 35. In this graph the input signal is SIG1IG. The two different outputs plotted here are those for a median filter of length 9 and LPF300 L9 α–trimmed eight elements, in effect an α–TL median filter. At first one may think

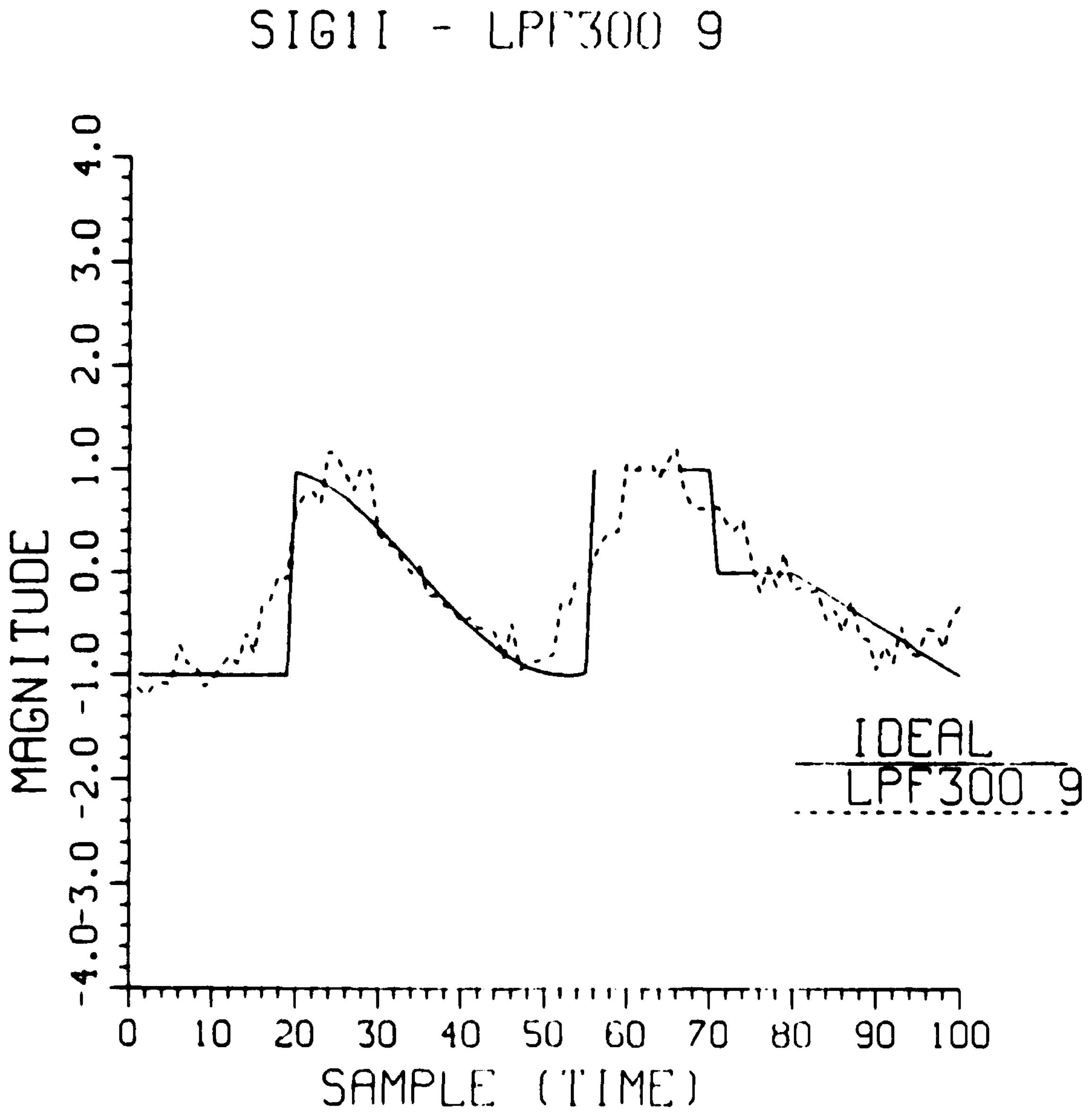


Figure 31 : SIG1I Filtered by LPF300 L9

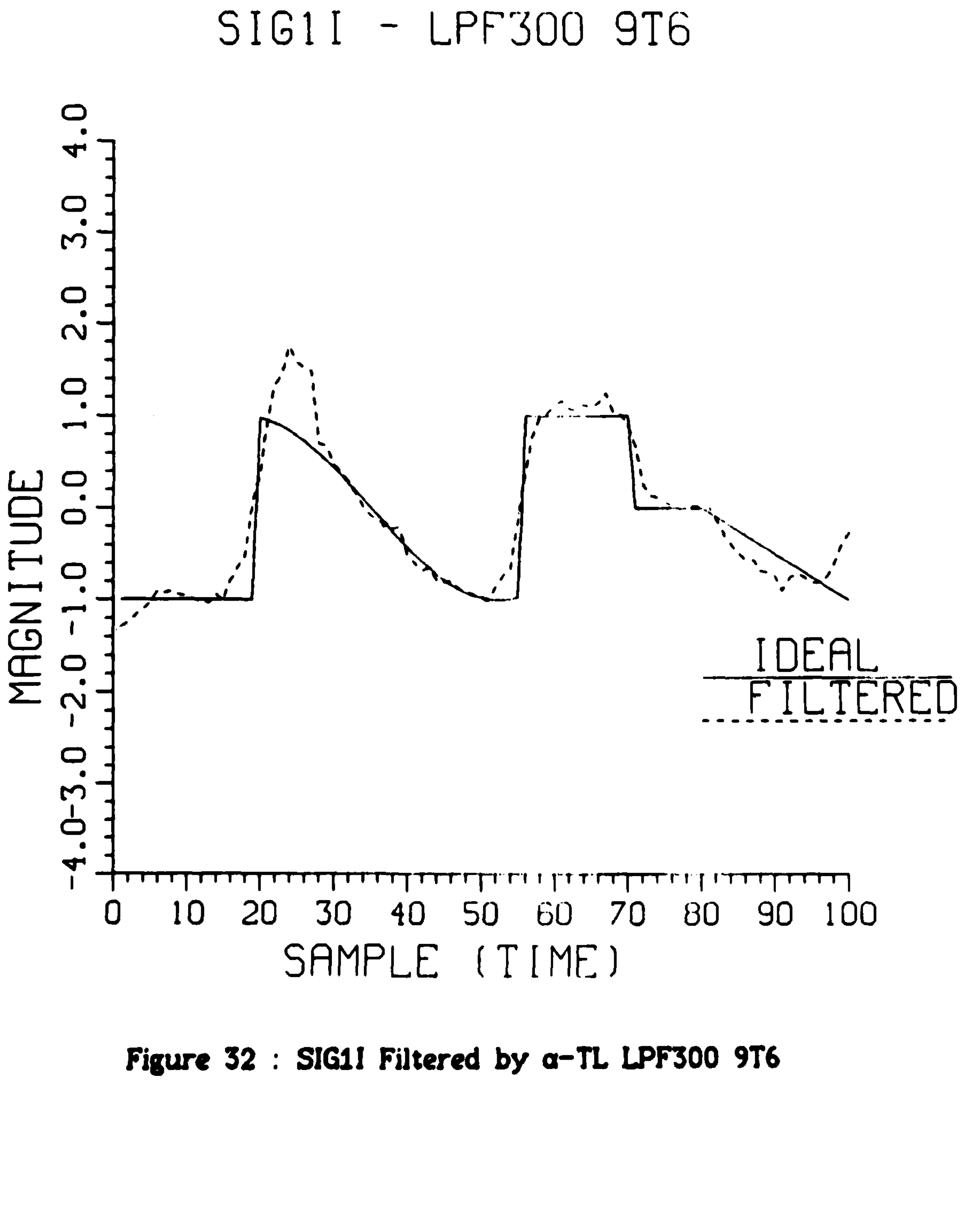


Figure 32 : SIG1I Filtered by α-TL LPF300 9T6

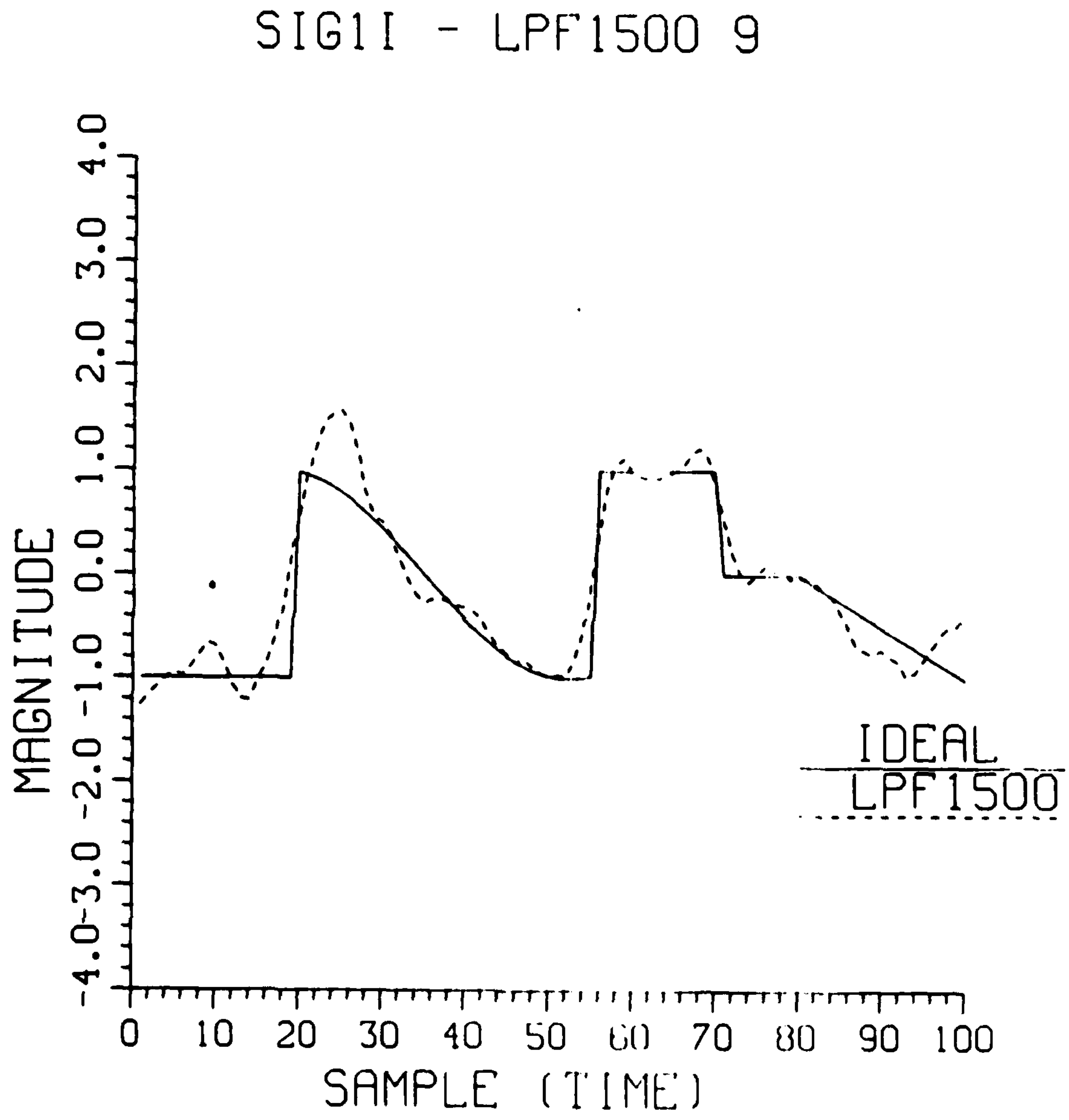


Figure 33 : SIG1I Filtered by LPF1500 L9

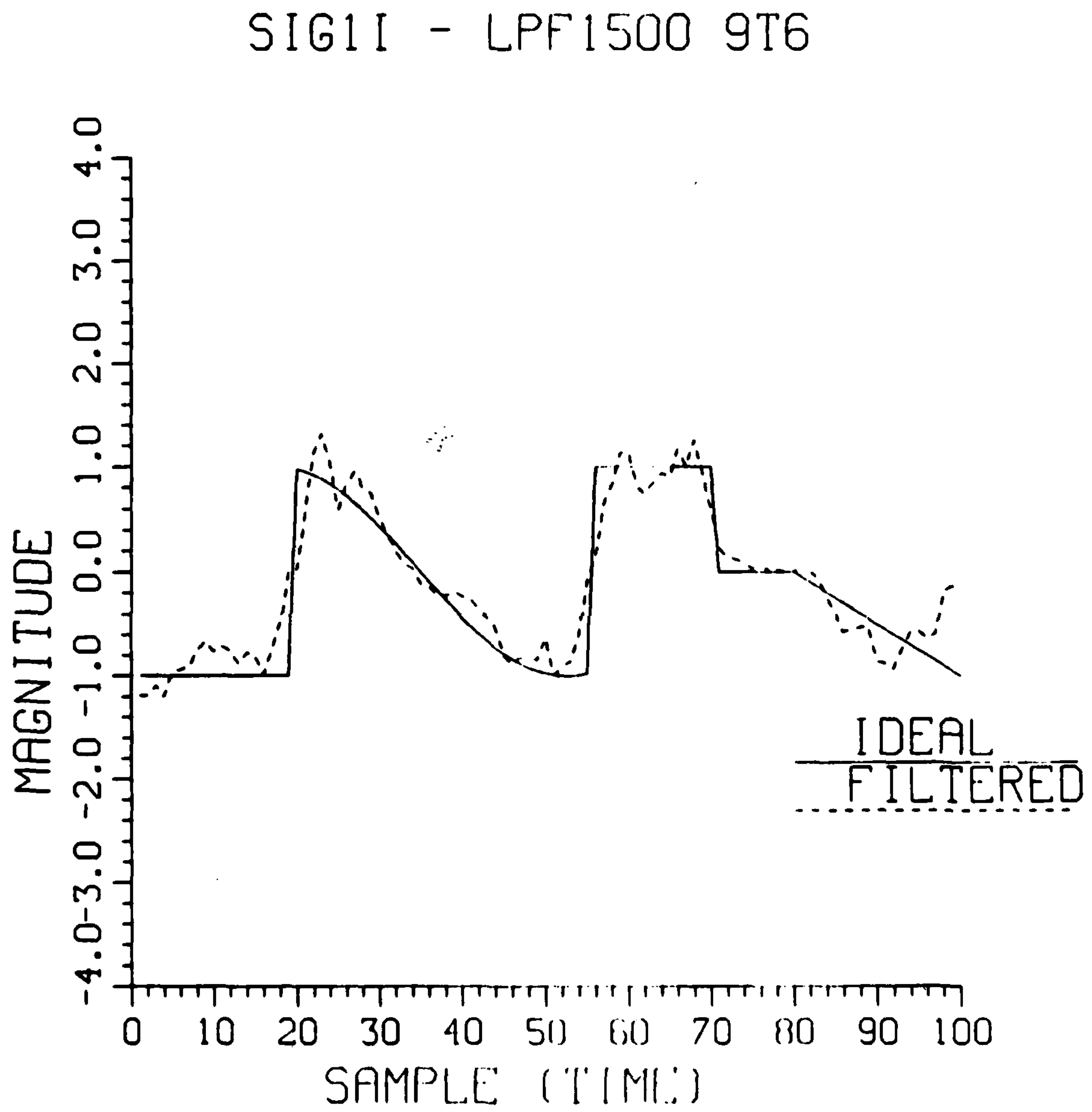


Figure 34 : SIG1I Filtered by α-TL LPF1500 9T6

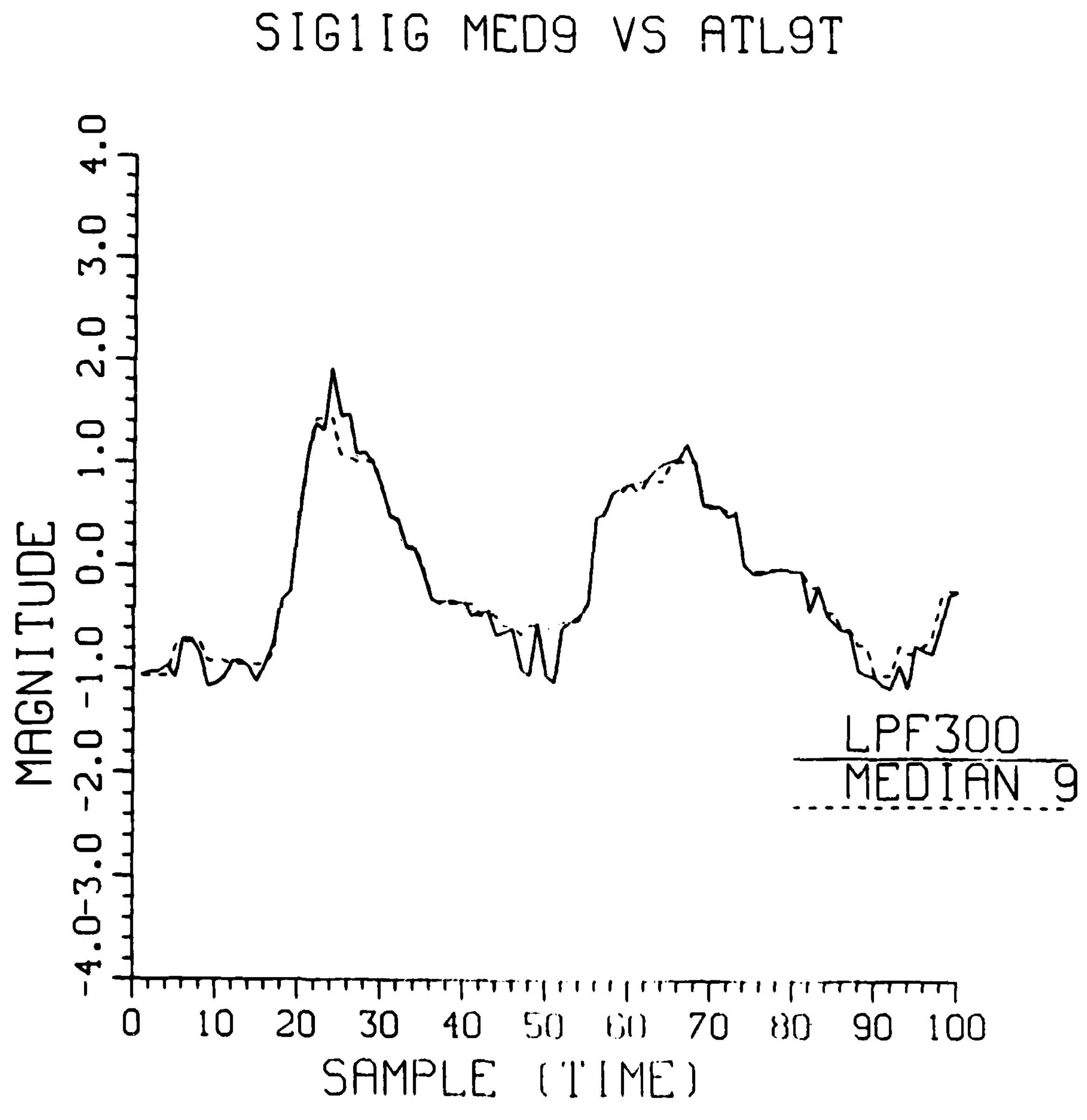


Figure 35 : Results of SIG1IG Filtered by Median 9 and α–TL LPF300 9T8

that these two filters should produce the same output since they are both nine element filters with eight elements trimmed. However, one must remember that the α–TL filter weights the input by the set of linear coefficients before the values are ordered and trimmed. So even though an element in the input window of an α–TL filter is the largest it is not necessarily going to be trimmed. It is only trimmed if its value times its weighting coefficient are large enough to force it to the ends of the window after sorting. As can be seen in the figure, the two graphs give the same outputs in some areas and are significantly different in others. The most notable feature is that the two filters produce nearly identical outputs in all areas of the graph where the signal is changing most rapidly: near each of the edges and along the center of the sinusoid. The areas where the outputs deviate the most are, in general, the flattest parts of the signal.

This suggests that something could possibly be done to improve the performance of the α–TL filter or at least suggests what the real differences are between the α–TL and α–TM filters. The reason for the poor performance in the flat areas of the signal is most likely due to the fact that the linear filter in this case has a finite band pass. We know that the mean filter is optimum in smoothing Gaussian noise, but the linear LPF must pass a certain band of frequencies. Note that the oscillations in these flat areas are not nearly so great as they were in the original signal. The high frequency noise has been filtered out, but some low frequency noise remains. The steep areas of the plot are changing fast enough that they are at the edge of the filter band, at least, thus the α−TL filter can do a good job of smoothing noise that has any higher frequency components in these relatively steep areas.

NMSE COMPARISON

A more effective or at least a more informative method of evaluating the performance of the α−TL filter in resisting outliers is with an error analysis between the filtered output signals and the original "ideal" signals. This discussion will concentrate on three different ideal signals each corrupted by either Gaussian noise, Laplacian (impulsive) noise, or an additive combination of both types of noise. The first signal is a low frequency signal with edges in it. It is the signal described above as SIG1 Following the same notation, a second signal, SIG2, is a sinusoidal signal with a frequency in the center of the pass band for two different band pass signals. SIG3 is a similar signal with a frequency that will place it in the pass band of both of the LP filters previously discussed. These three signals will allow several different comparisons and evaluations. Signals 1 and 3 will allow different filters to be compared in their performance with respect to signals that have edges versus slowly varying signals These two signals also give two examples for comparing the α−TL filter to the α−TM filter. Finally, signals 2 and 3 will show the relationship between the low pass and band pass cases of the α−TL filter

For every case tested two errors were measured normalized mean square error (NMSE) and normalized average error (NAE) These errors were calculated in the following manner

$$\text{NMSE} = (1/N)\ \frac{\sum [\,|y(n)| - |i(n)|\,]^2,}{\sum |i(n)|^2} \qquad n = 1,2,\ldots N \quad (9)$$

where y(n) is the filter output and i(n) is the ideal signal and

$$\text{NAE} = (1/N)\ \frac{\sum |y(n) - i(n)|,}{\sum |i(n)|} \qquad n = 1,2,\ldots,N. \quad (10)$$

Experimentally, as evidenced by Figures 36 and 37, NAE followed the same pattern as NMSE almost without exception. For this reason NMSE will be the only error discussed since it is the more common measure of error.

As evidenced by these two graphs, the α–TM filter performs better and better against Gaussian noise on a signal with edges in it as the number of elements trimmed is increased. This is largely due to the reduction of distortion on the edges by the reduced number of elements in the averaging. This effect is so acute that the median filter has the minimum NMSE for each filter length.

This conclusion is backed up by looking at the NMSE plot of this same set of filters when the input was the smooth signal 3 corrupted by Gaussian noise as shown in Figure 38.

Note in this graph that the vertical axis is only a small portion of that of the previous one. Note also that trimming has no advantageous effect. In fact, the error between filter output and

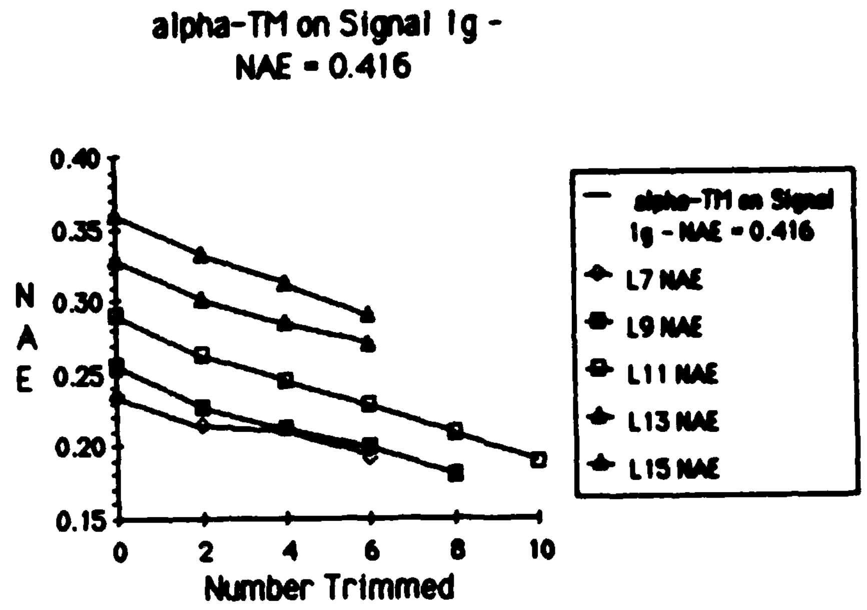


Figure 36 : SIG1G Filtered by α-TM - NAE

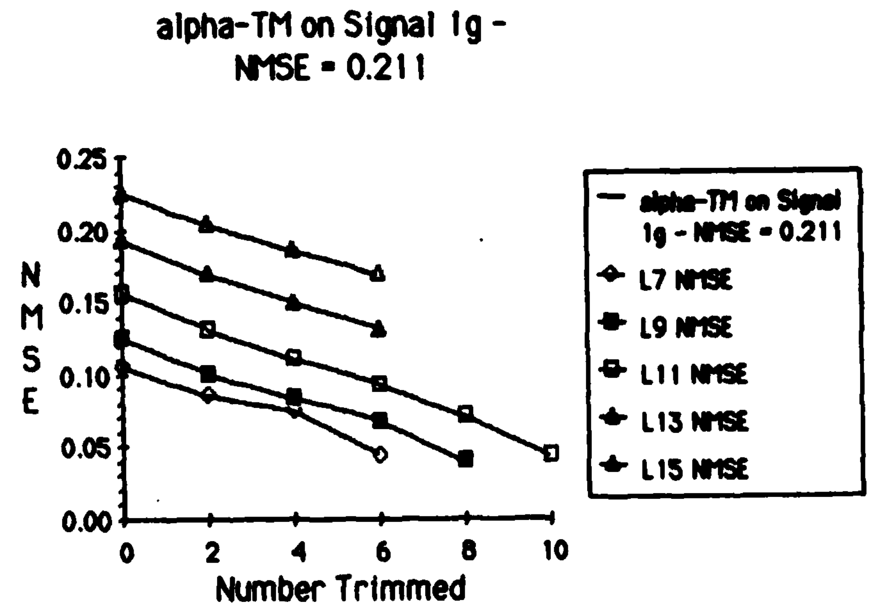


Figure 37 : SIG1G Filtered by α-TM - NMSE

the ideal signal actually increases somewhat due to the decreased number of elements used to smooth the noise and the inferior ability of the median filter to smooth Gaussian noise.

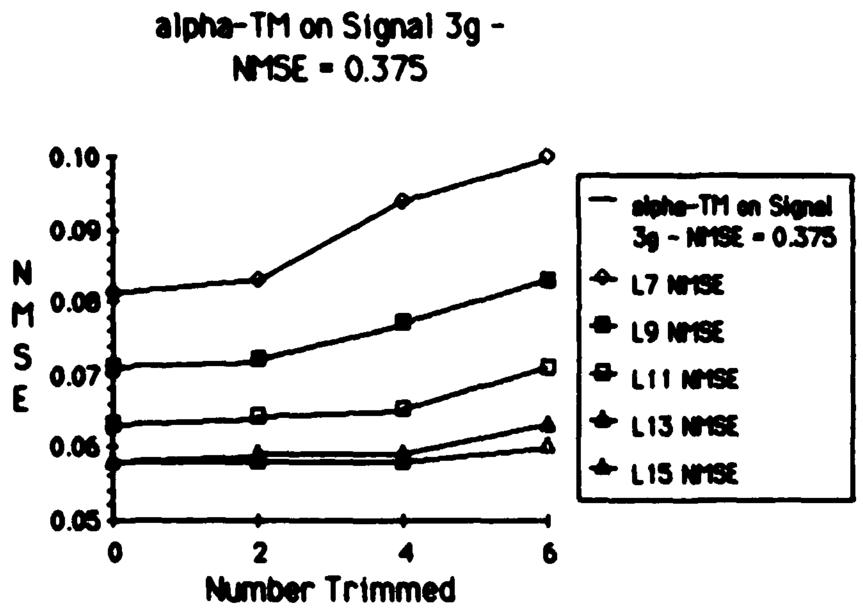


Figure 38 : SIG3G Filtered by α–TM – NMSE

The questions to ask now are 1) can the α–TL filters perform as well in smoothing Gaussian noise as the α–TM; and 2) how do the two compare in the impulsive noise case? Figures 39 and 40 show some interesting results. In the case of SIG1G the LPF300 α–TL filters (hollow point markers) perform like the α–TM filters shown in Figures 36 and 37. The distortion introduced by the linear filters by themselves is gradually reduced by more and more trimming. Whereas the LPF1500 filters (solid point markers) are able to follow the edges of the signal 1 fairly well initially due to

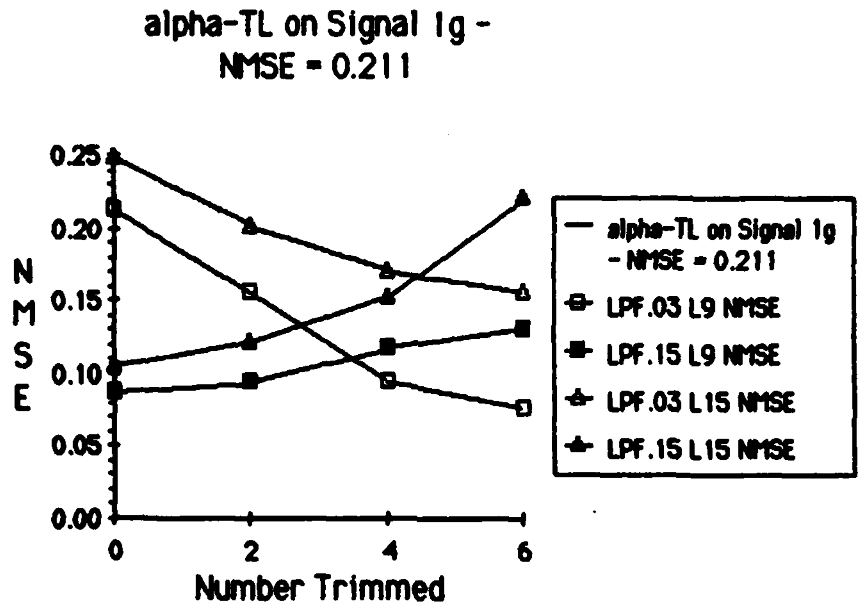


Figure 39 : SIG1G Filtered by α–TL – NMSE

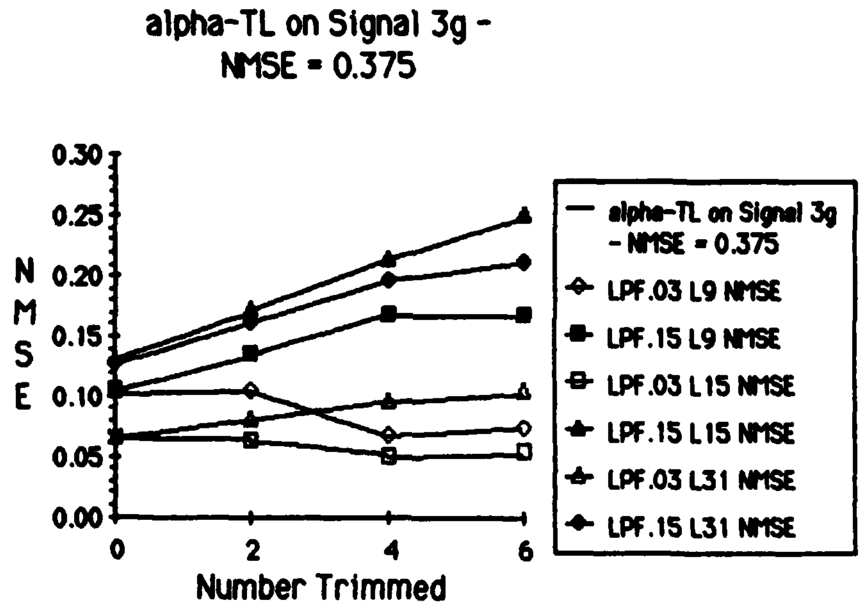


Figure 40 : SIG3G Filtered by α–TL – NMSE

the larger pass band, but trimming causes a degradation in noise smoothing ability and thus an increase in the NMSE. For the smooth SIG3G the results are turned in favor of the LPF300 filters' ability to block out more noise than the wider pass band of the LPF1500 filters. Indeed, the LPF300 filters perform nearly as well as the α–TM filters. It is interesting to note the unexpected decrease in error for α–TL LPF300 L9 in going from T=2 to T=4 and to a lesser extent the same thing happening to LPF300 L15 in Figure 40. Perhaps some sort of resonance is touched in the filter to cause this.

The α–TM filter and the α–TL filter perform nearly in the same manner in the presence of impulsive noise on signal 1 as in the presence of Gaussian noise. This is most likely due to the number of edges in signal 1. The filters that can best deal with the edges have the lowest NMSE values. The same characteristics which allow these filters to handle the edges in the signal are also the same characteristics that help the filters deal with outliers. An interesting point to note here is that LPF300 filters have the desired performance benefits derived from trimming elements off of the filter (NMSE goes down as T goes up), while the LPF1500 filters have a better initial starting point in both the impulsive and Gaussian noise cases. It would be interesting to design a filter with the steep rolloff of LPF300 having the pass band of LPF1500 to see if a better performing α–TL filter could be obtained.

The results for the performance of all of the filters on signal 3 in the presence of impulsive noise are somewhat surprising. The initial NMSE between SIG3I and the ideal signal before filtering is 0.504, while for SIG3G the NMSE was only 0.375. Yet, nearly all of the filters more effectively suppressed the impulsive noise than the Gaussian noise as can be seen when Figures 41 and 42 are compared to the figures previously shown for SIG1G. Note the different scales on the vertical axes. Notice also in this case that, in the presence of impulsive noise, trimming only has a beneficial effect for filters of short length and that the median is not the best filter to use on this smooth signal.

The error calculations for all of the filters were predictable for the case of the two signals corrupted by Gaussian and impulsive noise at the same time. Superposition of performance in the two cases seemed to hold very well.

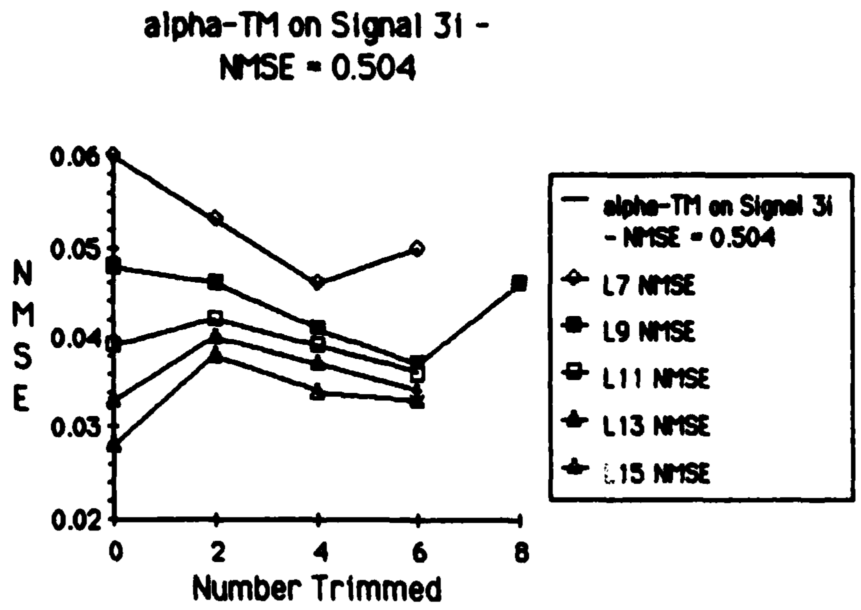


Figure 41 : SIG3I Filtered by α-TM - NMSE

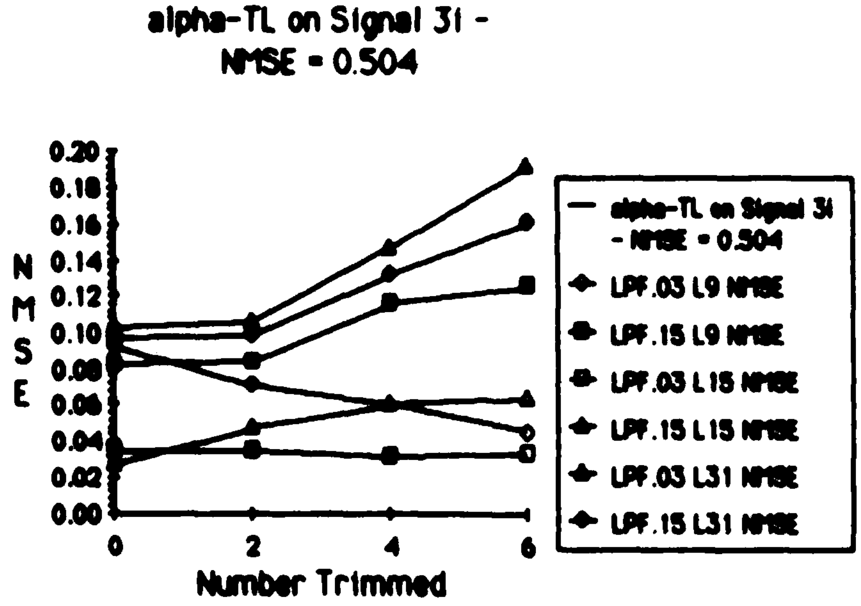


Figure 42 : SIG31 Filtered by α-TL - NMSE

The really interesting example in these error calculations is the case of signal 2 and the band pass filters. As was pointed out earlier, the ideal signal is just a sinusoid whose frequency lies in the pass band of the two band pass filters discussed in part A. As one can see in Figure 43 the linear filter performs an adequate job of smoothing Gaussian noise. Trimming elements off of either filter does not have a very big or very consistent effect. In general, α-trimming tends to help the shorter filters and harm the longer ones. This is probably due to the trimming introducing more distortion in the longer filters since the signal changes are short compared to the lengths of the longer filters.

However, when the signal is corrupted with impulsive noise, α-trimming can have a great effect as evidenced in Figure 44. For

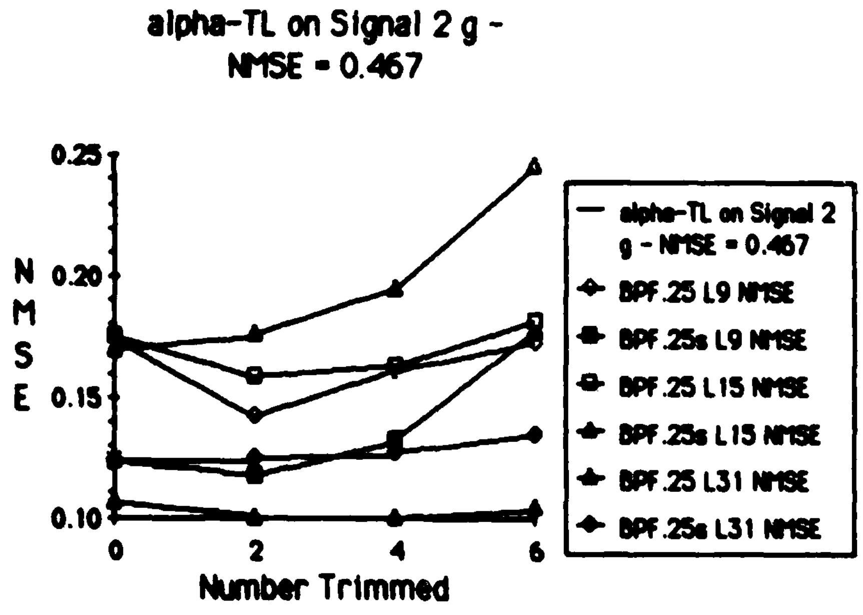


Figure 43 : SIG2G Filtered by α–TL – NMSE

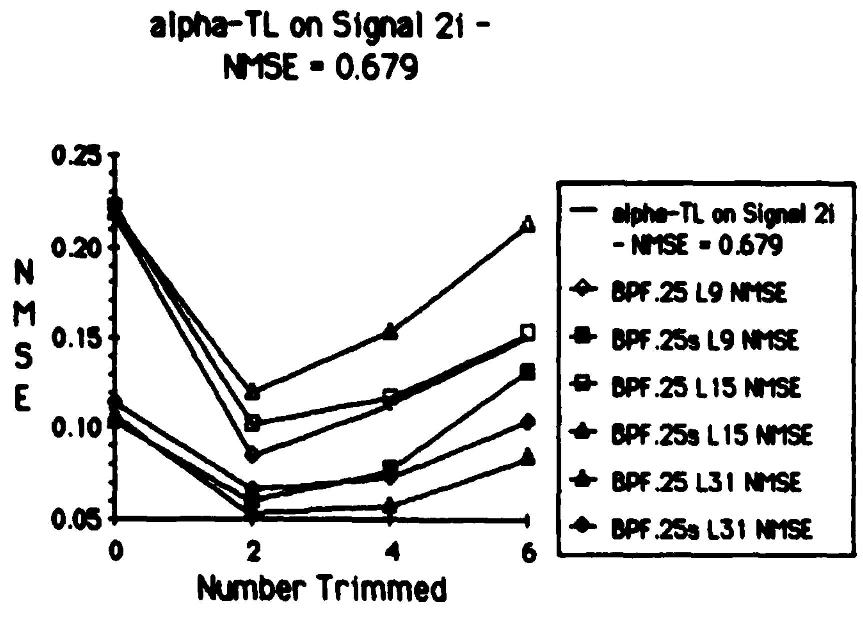


Figure 44 : SIG2I Filtered by α–TL – NMSE

this signal, the kind of affect the α-TL filter was supposed to have is finally realized. In this instance, the choice of an α-trimmed linear filter clearly has an advantage over its corresponding linear filter, especially for only two or four elements being trimmed. As one might expect, when signal 2 is corrupted by both Gaussian and impulsive noise it maintains the basic performance shown in the previous figure. The α-trimming helps reject the impulses and thereby decreases the NMSE without sacrificing very much in the way of Gaussian noise suppression.

References

[1] Yong Hoon Lee and Saleem A. Kassam, "Generalized Median Filtering and Related Nonlinear Filtering Techniques," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 3, Jun 1985, 680.

VI. SUMMARY, FUTURE RESEARCH AND CONCLUSIONS

A. SUMMARY

The α-trimmed linear filter model came out of a natural development of evolutionary filter designs. It is another step in the process of trying to marry desirable traits from different families of filters. It is noteworthy in that it is a first attempt at combining the inherently nonlinear filter characteristic of outlier resistance with the definitely linear filter characteristic of frequency selectivity. A viable filter design having these two primary characteristics could be extremely useful.

The model has a solid foundation under it. Like the mean and median filters are a subset of the L filter, so are the L filter and the α-trimmed linear filter special cases of the generalized order statistic model outlined in Section III. As with any new design, however, there are problems to be overcome. Indeterminate gain for some of the filters and a serious degradation in frequency response characteristics for nearly all the examples considered here are two of the major things that need to be investigated further if the α-TL is to be a viable filter design. As this simulation showed, too much performance is given up in the frequency response of the linear filter for an amount of outlier resistance already achieved or exceeded in other filter designs. There may be some specific applications where an α-TL filter would be the best choice, but it has not proven worthy as a general filter model, _yet_.

B. FUTURE RESEARCH

OPTIMIZING THE COEFFICIENTS

Perhaps the most remarkable result of the entire simulation was the appearance that some filters were affected less by the α-trimming process due to their having a steep rolloff design in the transition band. This leads to the conclusion that some characteristic of the linear filter coefficients can be exploited in order to minimize the effects that α-trimming has upon the filter. No conclusive evidence was found in this simulation that actually pinned down the characteristic or characteristics of the coefficients that seemed to be resistant to α-trimming. An area for further study could include a way to optimize the linear filter coefficients so that this effect may be achieved. In addition, it may be possible to adaptively modify the coefficients to optimize their performance.

IIR FILTERING

Just as an IIR filter of a certain length can easily outperform an FIR filter of the same length, perhaps using a feedback loop in the generalized order statistic filter model could improve the frequency performance of the trimmed linear filter. Using feedback, the IIR linear filter can be designed to have a very steep rolloff. It has already been shown that some relationship exists between the steepness of the designed rolloff and the immunity of a filter to α-trimming. It is possible that an IIR filter may prove to be more resistant to the damaging effects of α-trimming. This may be the only fix necessary to adequately preserve enough stop band attenuation during the trimming.

The GOS filter model would have to be modified slightly in order to accommodate an IIR filter. A feedback loop on the time weighting coefficients is obviously necessary in order to have an IIR filtering operation.

MODIFIED TRIMMED LINEAR FILTER

Looking at the suggested modification outlined above makes one wonder what would happen if there were a feedback loop on the rank weighting coefficients (b_n's). This modification was made to the α-trimmed mean filter as outlined in Section II. The modification was called the modified trimmed mean (MTM) filter and performed at least as well as the α-TM filter.

Recall that the MTM filter provided a feedback loop that trimmed a number of elements from the active window based on a range parameter q. The median, m_k, was selected from the window and then all elements outside of the range $[\, m_k \pm q]$ were trimmed off. Having the number of elements trimmed become dependent on the data allowed the filter to act as a median filter when much of the data fell outside the range and like a mean filter when there was very little deviation from the median. The range parameter was chosen based on *a priori* knowledge of the noise.

This scheme may prove particularly useful in the case of the trimmed linear filter. A modified trimmed linear (MTL) filter could outperform an α-TL filter for two reasons. As the simulation showed, the major degradation in performance occurred when just two elements were trimmed. If the signal were particularly noisy with many impulses, the degradation would not be significantly greater than for a lower fixed number of samples trimmed. However, the real payoff would occur during the times when no elements were trimmed. The MTL filter would allow for the possibility of not trimming any elements during portions of the signal that did not require trimming. This would allow the full effect of the linear filter to be felt.

By choosing q such that an element was trimmed for only extreme cases, a filter may be developed that has some resistance to outliers while maintaining a good frequency response. This case would be extremely interesting to pursue. It may give an insight to just how sensitive a linear filter is to trimming of any kind.

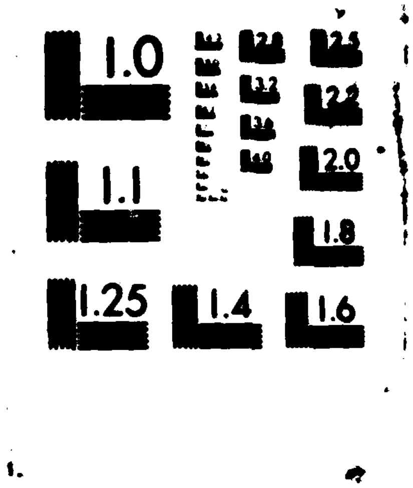
C. CONCLUSIONS

As can be seen from the development of this model and hinted at by some of the results of the simulation, the trimmed linear filter holds some promise in living up to its expectations as a frequency selective filter with outlier resistance properties However, as the simulation also showed, the α-trimming process can extract a fairly high price from the frequency performance of a linear FIR filter for a little resistance to outliers

This is the major drawback of the α-TL filter design. The section on outlier resistance showed that this filter design could have a comparable degree of impulse rejection to the α-trimmed mean filter in the low pass case. It at least did a fair job at rejecting outliers in all of the cases considered. In addition the results were surprisingly good for the BPF case. Trimming two or four elements from the BPF designs in the simulation produced a marked improvement in outlier resistance in the NMSE calculations. Using a more sophisticated trimming scheme may give even better rejection of impulses.

As the possibilities discussed in section B above show, there is still hope for the α-TL filter. The simulation showed several interesting relationships between some of the filter design parameters and how the α-trimming process affected the performance of the linear filters. Some of these may prove useful in helping to improve the performance of the α-TL filter to the point that it is a useful design for more than a few specific applications.

AD-A186 125   A GENERALIZATION OF ORDER STATISTIC FILTERS: THE   2/2
ALPHA-TRIMMED LINEAR FILTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH   D B RIDER 1987
UNCLASSIFIED   AFIT/CI/NR-87-101T   F/G 12/3   NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5

BIBLIOGRAPHY

Bovik, Alan C., Thomas S. Huang and David C. Munson, Jr. "A Generalization of Median Filtering Using Linear Combinations of Order Statistics." IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-31, No. 6, Dec 1983, 1342-50.

Lee, Yong Hoon and Saleem A. Kassam, "Generalized Median Filtering and Related Nonlinear Filtering Techniques." IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 3, Jun 1985, 672-83.

APPENDIX A

CALCULATION OF TRANSFER FUNCTIONS

This appendix contains some of the specific details used in calculating the transfer functions for the simulation. Several aspects of the model and computer system require some detailed explanation for those readers interested in the actual development and implementation of the model. The details described in this appendix are 1) Gaussian and Laplacian random number generation; 2) the histogram of calculated data and its implications; 3) window selection for the input data; and 4) error calculation to give an idea of confidence in the simulation.

RANDOM NUMBER GENERATION

The Pascal compiler used on the IBM 4341 system provided a random number generation function that was uniformly distributed. In the GOSF simulation it was necessary to generate both Gaussian and Laplacian random variables. This was done using a transformation of variables.

In order to transform variables from one distribution to another it is necessary to set the probabilities of the desired variable equal to those of the known variable as shown below for the case of a known uniform rv and the desired Gaussian rv:

$$\int_{0}^{u} dx = \int_{-n}^{n} \exp\{-y^2/2\sigma^2\}dy/\sqrt{2\pi}\sigma \qquad \text{(A1)}$$

where u is a uniform rv and n is a Guassian rv with standard deviation σ. However, we know from basic probability theory class that the normal distribution cannot be solved in this form.

Therefore, instead of solving the above equation we solve

$$\int_0^u dx = \int_0^r (y/\sigma 2)\exp\{-y^2/2\sigma^2\}dy \tag{A2}$$

where u is the same uniform rv, but r is now a Rayleigh rv. This equation can be manipulated and solved for r as a function of u. The Rayleigh distribution is obtained by looking at a two dimensional space that is Gaussian along x and y. The radial variable of this space has a Rayleigh distribution and the angle variable is uniformly distributed. Transforming the known uniformly distributed variable into a Rayleigh and generating an additional uniform rv will allow an indirect transformation to two Gaussian random variables. The above equation is solved for r in terms of u by substituting variables, integrating and solving. The final equation becomes

$$r = \sqrt{-2\sigma^2\ln(1-u)}. \tag{A3}$$

Then combining this Rayleigh rv with a uniformly distributed rv θ we make the two independent Guassian random variables

$$x = r\cos(\theta) \tag{A4}$$

$$\text{and}\quad y = r\sin(\theta). \tag{A5}$$

The same approach is taken for the Laplacian (or double-exponential) case, only equating the probabilities is directly solvable. The equation is

$$\int_0^u dx = \int_{-d}^{d} (\sqrt{2}/2\sigma)\exp\{-(\sqrt{2}/\sigma)|y|\}dy \qquad \text{(A6)}$$

where u is the uniform rv and d is a Laplacian rv with standard deviation σ. This equation is solved by dividing the right integral into two parts to get rid of the absolute value of y, integrating and solving. The result gives

$$d = -(\sigma/\sqrt{2})\ln(1-u). \qquad \text{(A7)}$$

The results for either case can be quickly checked by noting that as $u\rightarrow1$ then $r\rightarrow\infty$ and $d\rightarrow\infty$. Likewise, as u approaches 0 then $r\rightarrow0$ and $d\rightarrow0$.

A test case was run for the Gaussian distribution. Twenty thousand uniform random variables were generated and were transformed to two sets of ten thousand Gaussian random variables. The results including the expected Gaussian percentages are shown in Table A1.

Table A1: Random Number Transformation

x/σ	#uniform	Cumulative #Gaussian 1	Cumulative #Gaussian 2	Cumulative Normal %
0.5	2077	3875	3882	0.3830
1.0	2000	6834	6881	0.6826
1.5	1948	8646	8655	0.8664
2.0	1905	9539	9526	0.9544
2.5	1930	9866	9860	0.9876
3.0	2080	9978	9968	0.9974
3.5	2038	9992	9994	0.9996
4.0	2044	9999	10000	0.9999
4.5	1979	10000	10000	1.0000
5.0	1999	10000	10000	1.0000
Total	20000			

WINDOW SELECTION

One of the most difficult problems to solve in setting up the simulation was the method for calculating the transfer functions for the α-trimmed linear filter. As was explained in the Chapter IV section on transfer functions, it is not a simple matter to find the frequency response characteristics of a nonlinear filter that is designed to reject impulses on the input signal. An attempt to find a simple deterministic method eluded all efforts. For this reason a probabilistic measure was attempted.

The first problem encountered in using Gaussian noise as the filter input was one of leakage. The transfer functions were adversely affected by noise near the ends of the data stream. This problem was easily alleviated by windowing the input data. Several different windows were tried. The Blackman window was chosen based on its performance in calculating the transfer function of a linear filter with none of the elements trimmed. By visual inspection it was determined that by using the Blackman window, given by

$$w(n) = 0.42 - 0.5\cos(2\pi n/(N-1)) + 0.08\cos(4\pi n/(N-1)) \quad \text{(A8)}$$

for $0 \leq n \leq N-1$, the linear filter frequency response was most closely determined with the random input transfer function calculation. In fact, as was discussed in Chapter IV, the NMSE would go to zero when using just two trials of random data input. This is due to the fact that, of the windows chosen, the Blackman window had the best leakage characteristics. One needs to take into consideration, however, that in the determination of the transfer functions for the α-TL filters the relatively poor resolution characteristics of the Blackman window function may adversely affect the results, but cannot be avoided.

HISTOGRAM DISTRIBUTION

A somewhat more interesting problem came about in the actual calculation of the transfer function. All of the data about transfer functions presented in this thesis is based on 500 inputs of 256 samples of white Gaussian noise. Table A1 in the previous section showed how well the noise fit the Gaussian distribution and Figure A1 shows the autocorrelation of 2048 samples of this noise. The correlation length indeed appears to be very near zero.

However, the first attempt to calculate the transfer function of a particular filter from the 500 trials was done by averaging each

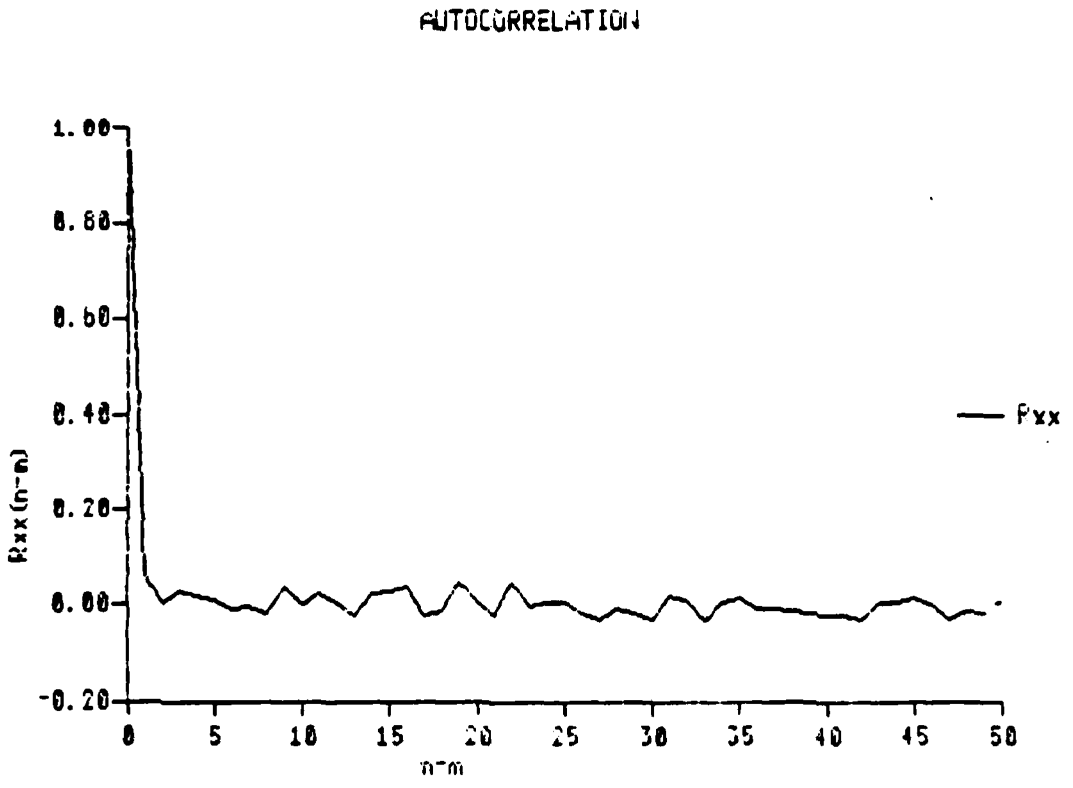


Figure A1 : Autocorrelation of 2048 Gaussian Samples

of the 500 transfer functions on a point by point basis. This procedure produced transfer functions which were unexpectedly "noisy" in nature. That is to say, even though the basic shape of a given transfer function was believable, there were unexplained bumps and spikes on top of the shape, all of which were positive. Correcting a small error in the FFT algorithm and fixing a problem with the number of significant digits in the data generation program only alleviated a small amount of the apparent noise on the basic transfer function shape.

An amount of uncertainty in the calculation was to be expected since the transfer functions were generated using a random input. Indeed, a small random deviation from a smooth shape was observed. However, on top of this fluctuation were unexplained spikes at various frequencies of the transfer function. Initially this was explained as a frequency selectivity of the α-trimming process or perhaps some sort of resonance was introduced by the trimming. But, as was explained in Chapter V, the α-TL filter was later found to not have any irregular frequency responses.

The answer was found when histograms of transfer function values at given points of the function were generated. This showed that the distribution of values at each point was not very symmetric over much of the transfer function. Only in the frequencies of the pass band of the filter was the histogram anywhere near symmetric. For most of the rest of the points the histogram had the majority of the points grouped around some relatively small value with an exponentially shaped fall off of points away from this main hump. This led to the conclusion that averaging the 500 values may not be the best way to show the central tendency of all of the values at each point of the transfer function. This conclusion was further supported by more investigation into the histograms of the data which found some extremely large, unexpected values.

Figure A2 shows a transfer function for a band pass filter calculated early in the simulation (the function is symmetric about 0 Hz at sample 128) using point by point averaging. The arrows point to individual values of the averaged transfer function and identify which sample they are. One can see that the general shape of this transfer function is the same as those for some of the band pass filters in Chapter V with some impulsive-type noise added to it. Figures A3 through A6 show some representative histograms of the data. Specifically the histograms are for points 91 though 94, respectively. The vertical scale shows number of occurrences in a particular bin with the horizontal scale being ten times the actual value of the transfer function at that point for that trial. (note: the transfer function in Figure A2 was scaled by a constant after the histograms were generated; therefore, the average of all of the points of the histogram divided by ten will not necessarily give the value of magnitude shown in the graph of the transfer function.)

All of the histograms of the data in the stop bands are of this same basic shape. They show that the data is skewed to the right. Thus the central tendency of these distributions may be better

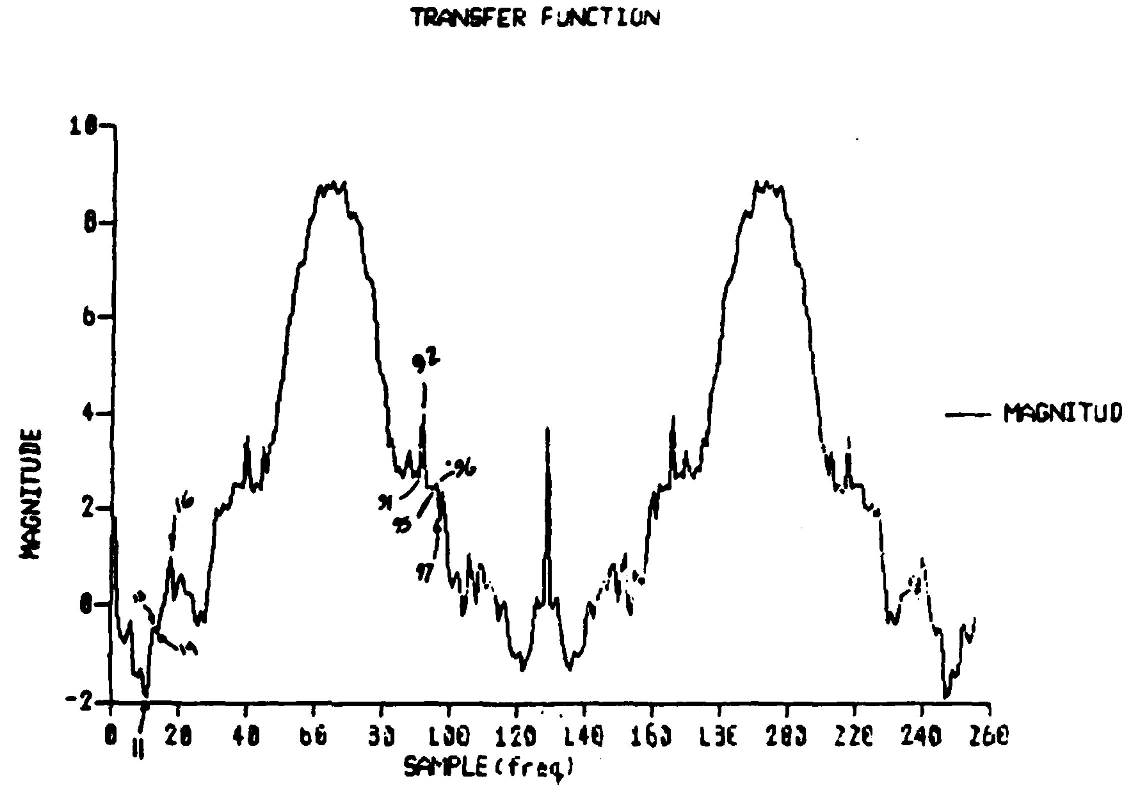


Figure A2: Transfer Function of BPF2500s L31 (linear scale)

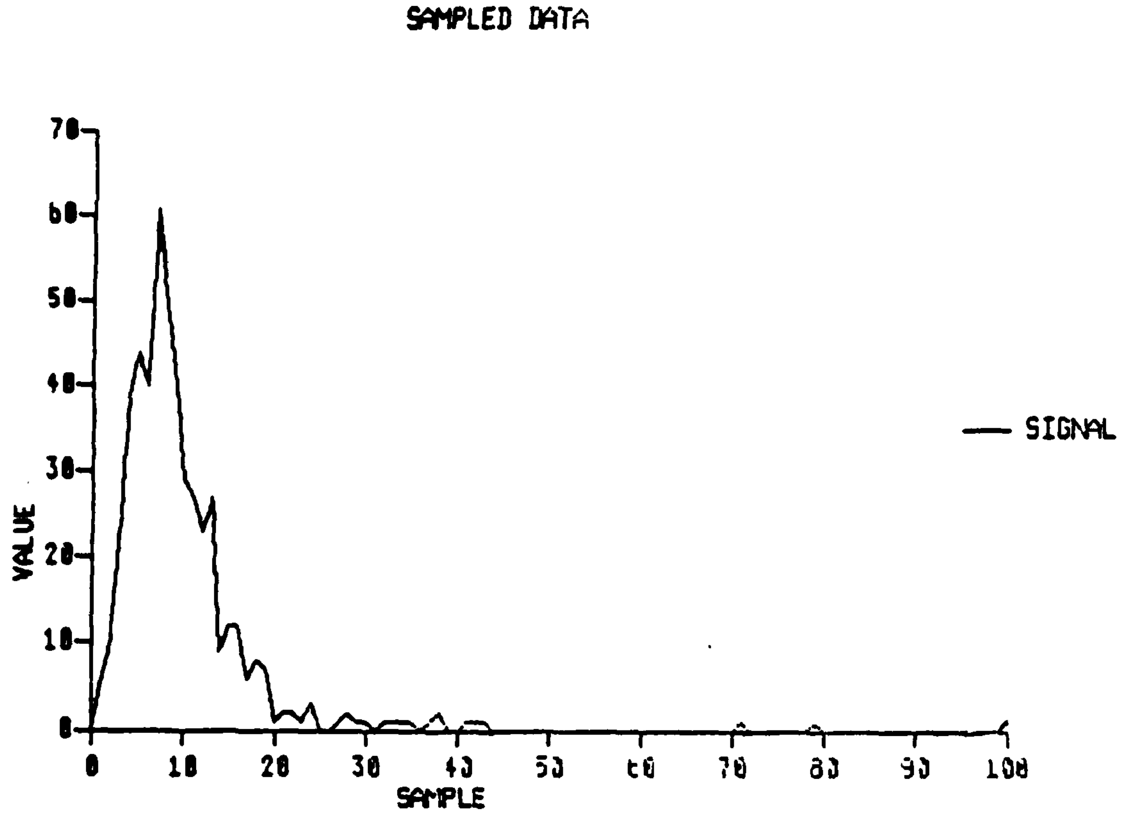


Figure A3: Histogram of 500 Values of Sample 91

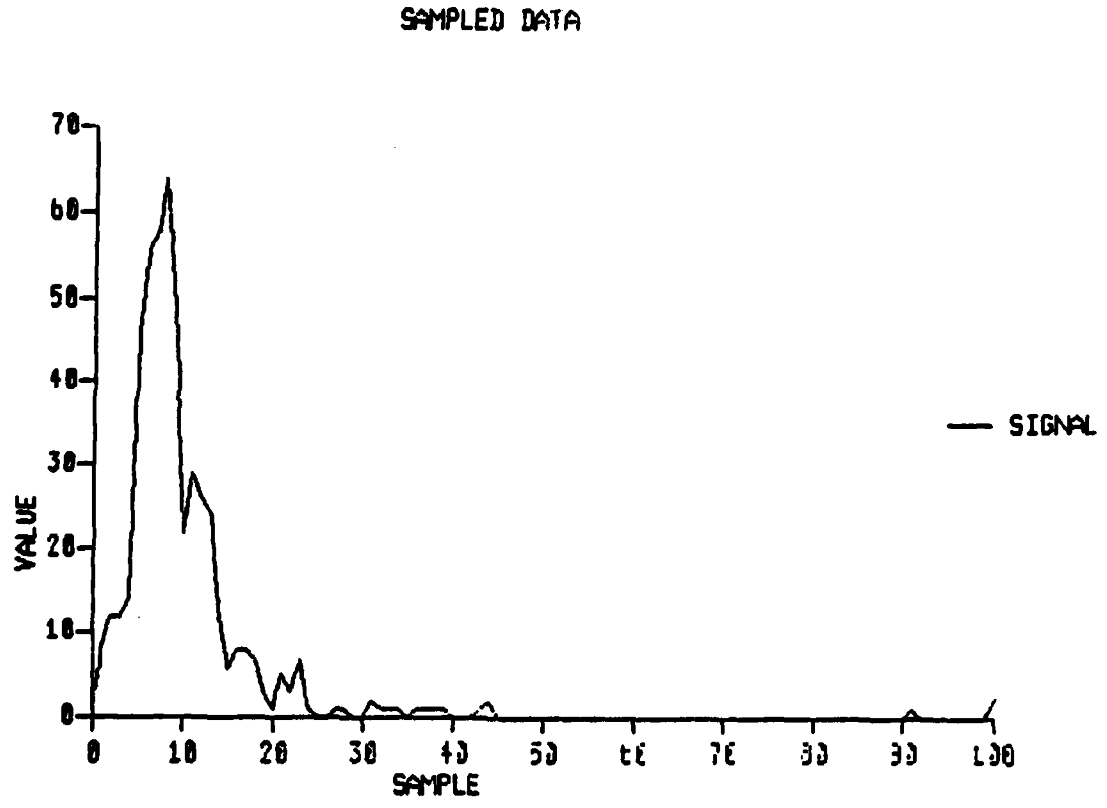


Figure A4: Histogram of 500 Values of Sample 92

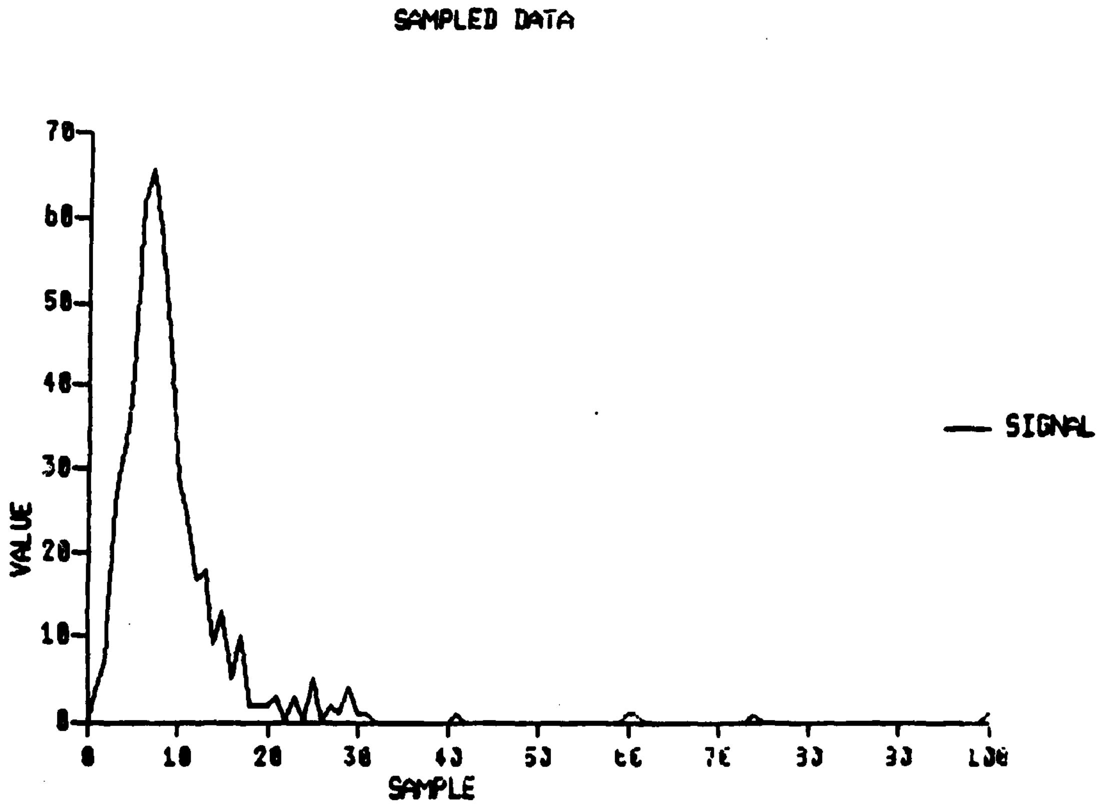


Figure A5: Histogram of 500 Values of Sample 93

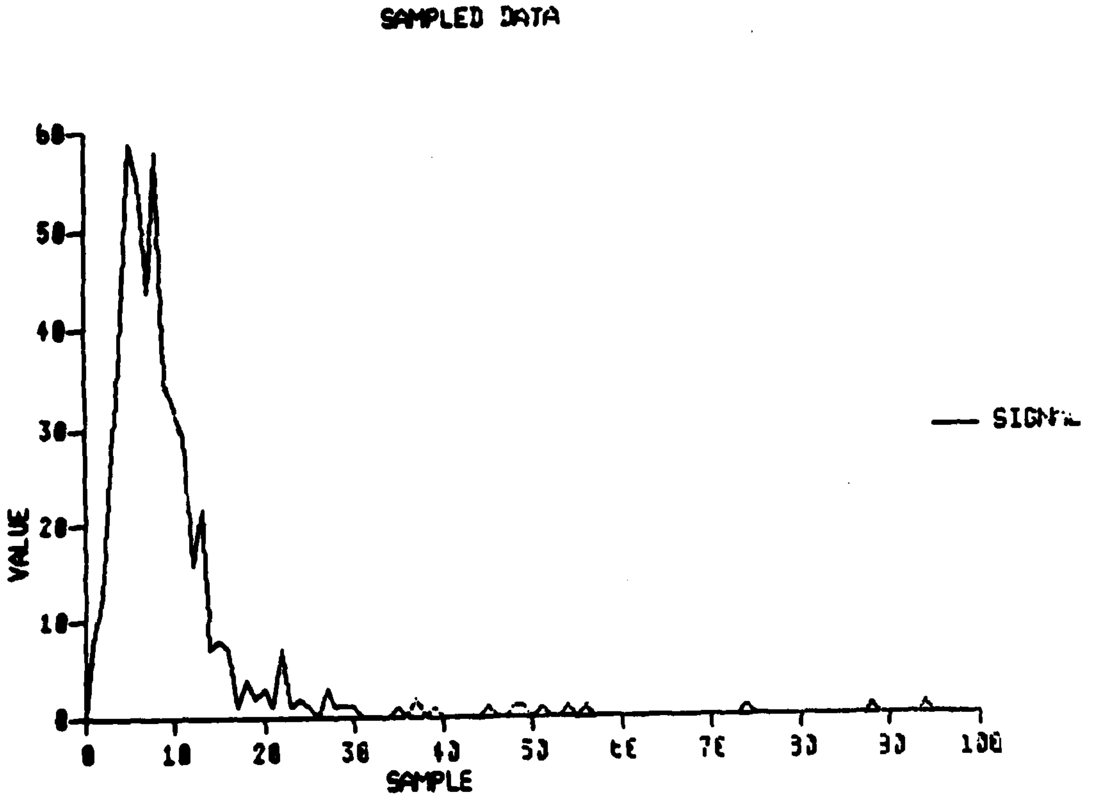


Figure A6: Histogram of 500 Values of Sample 94

described by the median rather than the mean. However, the real evidence for this conclusion comes from noticing the trials that fall into bin 100 in the data for samples 91, 92 and 93. The histogram generation program limited the value of any individual trial to 10. Any occurrence greater than 10 was put into bin 100. So the actual values for these large occurrences were printed out. For the single large occurrence in sample 91 the value was just 11.5. Likewise, for the single occurrence in sample 93 the value was 13.1. However, for sample 92 there were two large occurrences. One was only 11.5, but the second was 169.8, nearly 15 times greater than the next largest value! Since there were only 500 trials to average over this one extraordinary trial was able to add more than ten percent to the final averaged value at that particular sample.

This was the main reason for using the median of the data at each point to determine the "most likely" value for the transfer function. Yet this brings up another interesting point; what kind of certainty does this calculation provide? Is this simple approach to finding the transfer function actually valid in this case? These questions are discussed in the next section.

ERROR CALCULATION

The last two sections bring up some questions about the validity of the transfer function calculations used in this thesis. The windowing of the input data is necessary but forces a choice between some leakage/resolution trade-offs of various windows. The somewhat strange histograms of the calculated data along with the peppering of large outliers in this data introduces some additional uncertainty into the measurement of the transfer functions of the α–TL filters. The question is how far off are the measured transfer functions?

The choice of the window to use on the input data was straightforward. Of all the windows tried, the Blackman window did the best job at reducing the leakage problems encountered in the transfer function calculations. The different degrees of resolution between the various windows made no real visible difference under the criteria selected for choosing the best window. As stated before, the criteria was to match the impulse response frequency characteristics of a linear filter with the random input transfer function calculation of the same untrimmed linear filter. The Blackman window performed the best since it had the best leakage characteristics of all the windows tested. Once the leakage was reduced enough the transfer function matched almost identically with only one trial of random input to the expected frequency response of the linear filters. Therefore, it is assumed that it would be difficult to obtain better results for the transfer functions with respect to the window used.

There still may be some question in regard to the method of "averaging" the transfer function trials, however. In order to address this question some confidence intervals were found. During the median selection it was a relatively simple manner to choose other values at each point. Various "confidence intervals" were generated in this manner by selecting the 10 and 90 percent values to give an 80 percent confidence interval, for example. This process gave a picture like that shown in Figure A7 for the skinny band pass filter of 15 elements with 2 trimmed, BPF2500s 15T2. The figure shows the ideal transfer function of the linear filter and the transfer function of the filter with 2 elements trimmed selected by the median of 500 trials of Guassian noise as the input. The figure also shows 10 and 90 percent lines for the transfer function with 2 elements trimmed. These are the 50th and 450th largest values on a point by point basis of the ordered data from the 500 individual trials. They are normalized by the same constant as was calculated to normalize the median response to 0dB in the pass band. This gives a good idea of the range of values this method of calculating the transfer functions covers. One can see that if the 10 and 90 percent lines were properly normalized they would come fairly close to being identical to the median line. The only major difference is the wide variance at the dc point as was discussed in Chapter V.

A similar picture is shown in Figure A8 for the same filter with 6 elements trimmed. The major difference between the two figures is the expected increase in variance between the 10 and 90 percent lines for the filter with 6 elements trimmed. One may also note that it appears, by taking into account the possibility of getting the response of the 10 percent line in the pass band and that of the 90 percent line in the stop band, the frequency response for the filter may be nearly flat! In practice, however, one finds that this is not true. For an individual trial of Gaussian noise input the

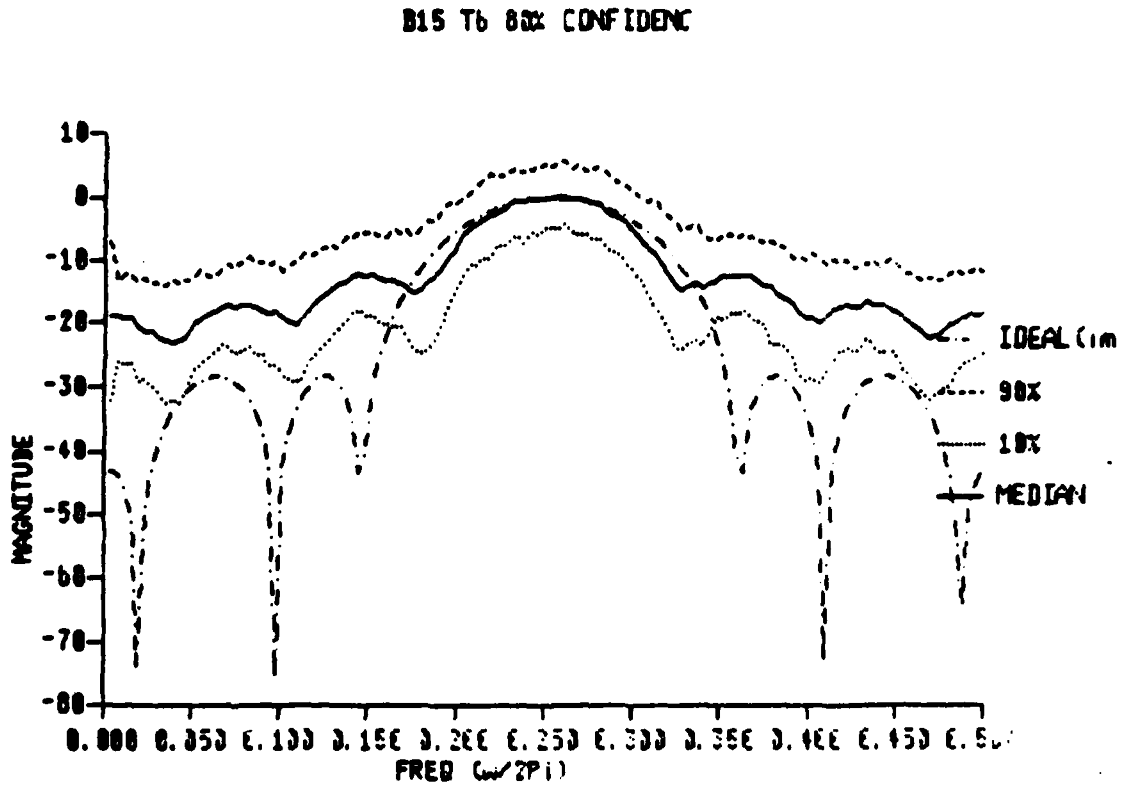


Figure A7: 80% Confidence Interval for BPF2500s 15T2

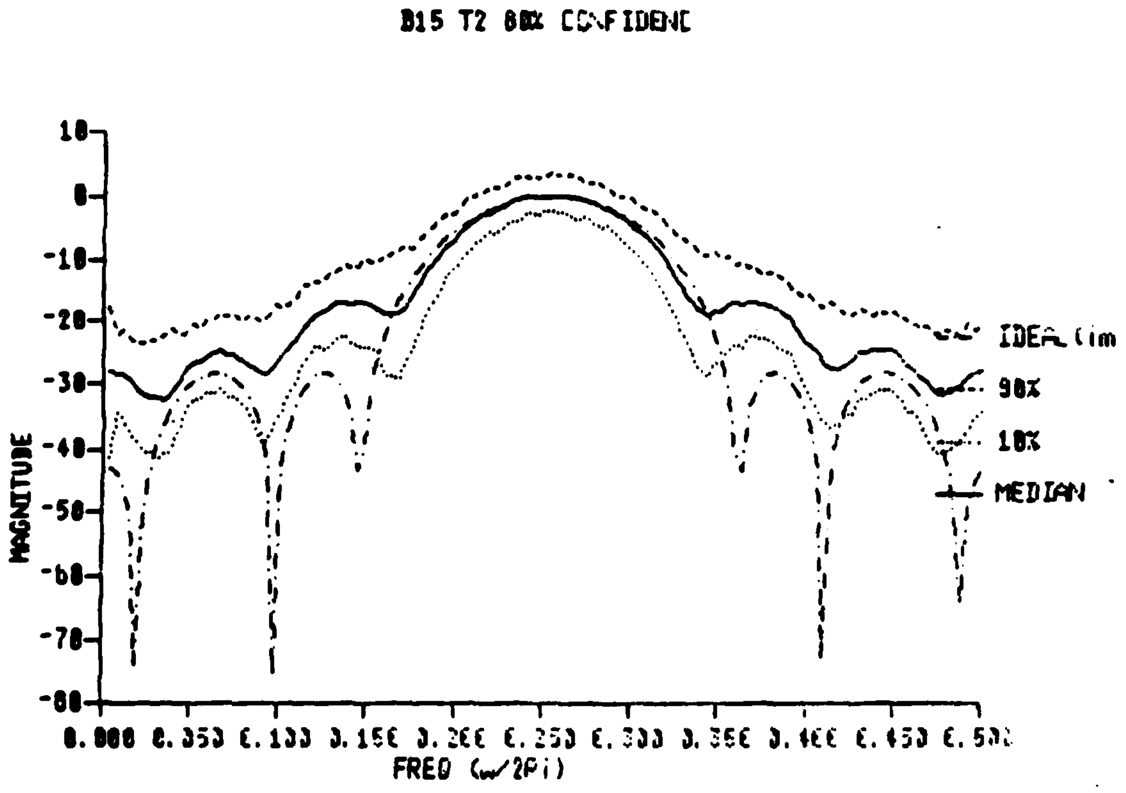


Figure A8: 80% Confidence Interval for BPF2500s 15T6

transfer function always maintained the same basic transfer function shape. The shape was simply much more "noisy" than the relatively smooth response given by the median of 500 trials. These confidence intervals merely show the magnitude of possible variations within a transfer function for a single input.

This explanation makes sense when one takes into account the nonlinearity of the α–TL filter. Just as for the simple median filter, it is not possible to predict what the exact frequency response of the filter will be for any arbitrary input. It can only be said that the filter will have a general frequency response most likely represented by the shape given by the median of 500 trials of random input as shown in this thesis. This is just the same as saying that the median filter is a low pass operation. The median's exact frequency response cannot be determined without knowing the input. It is only known in general that it will act in a low pass fashion with an approximate cutoff and somewhat variable stop band performance.

APPENDIX B

COMPUTER FILTER MODEL

In order to investigate the characteristics of the generalized trimmed linear filter I developed a program that would implement the various types of filters, a program to generate input signals to filter, and some programs to compare performances of the various filters on a set of input signals. The filter model program is simply an implementation of the model described in the previous section. All of the filters we are interested in using as comparisons are subsets of this generalized model.

LANGUAGE SELECTION AND MODEL STRUCTURE

I chose to write the program in Pascal instead of the perhaps more obvious choice of the more computationally efficient FORTRAN for two reasons. First was that the model structure seemed most logically implemented using pointers with a linked list architecture. FORTRAN does not have a pointer data type and, therefore, a linked list structure would be difficult to implement. Second, I like the stronger data type checking in Pascal or, rather, I dislike the "weak typing" of FORTRAN. This was an important consideration due to the moderate complexity of the model combined with the slight degree of complexity of the pointer/linked list structure to be implemented.

The reason I chose a linked list structure was that it most efficiently allowed me to add and delete elements from the active filter window and provided a relatively efficient sorting routine. Another good choice might have been to use an array structure for the windows, but I discarded this as less efficient in the two main manipulations of the windows: adding and deleting elements and sorting. The efficiency of the sorting routine becomes important in this model since the window must be re-sorted at each step through the input signal. The insertion sort routine for a linked list structure takes the first value and calls it the smallest value. Each successive value to be sorted is inserted into the proper place in the sequence taking care to keep track of the smallest value.

PROGRAM SET DESCRIPTION

The following is a brief description of each of the main component programs used in the simulation as well as a discussion as to how they were combined to form the large, repetitive program MEGAMEDIAN. The descriptions here are limited to the major Pascal programs. Many smaller usually machine dependent programs were written to produce graphical outputs and to better control the data file structure on the IBM 4341. Since these programs would be of little use to the general programmer they have been omitted here.

MAKEDATA: This program allows the user to generate an input sequence that is to be filtered. The input may have any combination of the following possible components: constant levels, ramps, steps or impulses, sinusoids or blocks of sinusoids, Gaussian noise, or Laplacian noise. In addition, the user controls many of the variables associated with each of these signal components including location in the signal, magnitude, duration, and variance, to name a few. The user also picks the number of samples in the signal. This value is stored as the first data point in the data file and is used by the filtering and Fourier transform programs to properly handle the signal. In addition, once the desired signal has been generated the user may store it in one of two different files so that one may be used as a reference while the other is varied. The user also has the option to go back to either of these two files in the future and add onto the existing signal.

FREQ: This program takes the discrete Fourier Transform (DFT) of the input. The DFT is implemented with the decimation-in-frequency fast Fourier transform (FFT) algorithm. The user is prompted for which data file to use as the source for the Fourier transform. The options are either of the two signal files created by MAKEDATA, the output of the autocorrelation program, R_{xx}, or the output of the filter program. If the input signal is not factorable by 2 then the remaining elements are padded with zeros. After the FFT is taken, the real and imaginary parts are converted to magnitude and phase since this simulation is primarily interested in the magnitude of the transform. The user is then given the option to have the output converted to a log scale. Finally, the user has several options to store the output in different files for later manipulations and calculations.

FILTER: This program obviously implements the filter model discussed in Chapter III. It is only important here to note the inputs the user may make. The user controls the window size to be used and then selects the type of filtering operation to perform. Once this choice has been made the program prompts the user for the necessary inputs to complete the filtering operation.

MSE: This program calculates not only the NMSE between two inputs, but also the NAE. In addition the program generates two output files to be graphed. The first merely combines the two inputs into a single file so that both graphs may be plotted at the same time. The second file is a graph of the difference signal between the two inputs. These two graphs are useful not only for simulation insights, but also for troubleshooting. The user has a wide range of choices to select which data files to calculate the errors between. The program can handle files stored in log magnitude as well.

MEGAMEDIAN: This file creates transfer functions. MAKEDATA, FREQ, and FILTER are all procedures in this very large program. The main program loops through these three programs as well as a procedure to calculate the transfer function for each of the 500 trials in order to generate a transfer function for a particular filter.

The listings for these five programs are Appendix D.

APPENDIX C

SET OF LINEAR FILTERS USED

Table C1 : Filter Coefficients for LPF300 L9

```
*************************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

                FILTER LENGTH =    9

                ***** IMPULSE RESPONSE *****
                     H( 1) =   .23328290E+00 = H(  9)
                     H( 2) =   .45690290E-01 = H(  8)
                     H( 3) =   .48455940E-01 = H(  7)
                     H( 4) =   .50179170E-01 = H(  6)
                     H( 5) =   .50768600E-01 = H(  5)

                         BAND  1          BAND  2
 LOWER BAND EDGE      .000000000       .050000000
 UPPER BAND EDGE      .010000000       .500000000
 DESIRED VALUE       1.000000000       .000000000
 WEIGHTING           2.000000000      1.000000000
 DEVIATION            .211253600       .422507300
 DEVIATION IN DB   -13.503900000     -7.483307000

 EXTREMA FREQUENCIES
     .0100000     .0500000     .1333333     .2533333     [illegible]
     .5000000

*************************************************************
```

Table C2 : Filter Coefficients for LPF300 L15

```
**********************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

              FILTER LENGTH =  15

              ***** IMPULSE RESPONSE *****
                   H( 1) =   .11714120E+00 = H( 15)
                   H( 2) =   .40280550E-01 = H( 14)
                   H( 3) =   .45621000E-01 = H( 13)
                   H( 4) =   .50370080E-01 = H( 12)
                   H( 5) =   .54310600E-01 = H( 11)
                   H( 6) =   .57236030E-01 = H( 10)
                   H( 7) =   .58985310E-01 = H(  9)
                   H( 8) =   .59580910E-01 = H(  8)

                        BAND  1          BAND  2
LOWER BAND EDGE      .000000000      .050000000
UPPER BAND EDGE      .010000000      .500000000
DESIRED VALUE       1.000000000      .000000000
WEIGHTING           1.500000000     1.000000000
DEVIATION            .131174300      .196761400
DEVIATION IN DB   -17.643010000   -14.121190000

EXTREMA FREQUENCIES
    .0100000    .0500000    .0851562    .1515625    .2179687
    .2882813    .3585938    .4289063    .5000000

*********************************************************
```

Table C3 : Filter Coefficients for LPF300 L31

```
****************************************************

                    FINITE IMPULSE RESPONSE (FIR)
                    LINEAR PHASE DIGITAL FILTER DESIGN
                    REMEZ EXCHANGE ALGORITHM

                    BANDPASS FILTER

             FILTER LENGTH =  31

             ***** IMPULSE RESPONSE *****
                  H( 1) =  -.11461630E-01 = H( 31)
                  H( 2) =   .14994630E-01 = H( 30)
                  H( 3) =   .14010050E-01 = H( 29)
                  H( 4) =   .16036500E-01 = H( 28)
                  H( 5) =   .19599590E-01 = H( 27)
                  H( 6) =   .23948930E-01 = H( 26)
                  H( 7) =   .28734110E-01 = H( 25)
                  H( 8) =   .33685870E-01 = H( 24)
                  H( 9) =   .38617050E-01 = H( 23)
                  H(10) =   .43318480E-01 = H( 22)
                  H(11) =   .47660100E-01 = H( 21)
                  H(12) =   .51456090E-01 = H( 20)
                  H(13) =   .54554890E-01 = H( 19)
                  H(14) =   .56882300E-01 = H( 18)
                  H(15) =   .58285860E-01 = H( 17)
                  H(16) =   .58740180E-01 = H( 16)

                             BAND  1         BAND  2
 LOWER BAND EDGE         .000000000       .050000000
 UPPER BAND EDGE         .010000000       .500000000
 DESIRED VALUE          1.000000000       .000000000
 WEIGHTING              1.000000000      1.000000000
 DEVIATION               .039385830       .039385830
 DEVIATION IN DB      -28.093170000    -28.093170000

 EXTREMA FREQUENCIES
     .0000000     .0100000     .0500000     .0636719     .0929687
     .1261719     .1593750     .1945312     .2277344     .2628906
     .2960938     .3312500     .3644531     .3976563     .4328125
     .4660156     .5000000

*****************************************************
```

Table C4 : Filter Coefficients for LPF300 L63

```
                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

                 FILTER LENGTH =  63

                 ***** IMPULSE RESPONSE *****
                      H( 1) =  -.18686920E-02 = H( 63)
                      H( 2) =  -.15462900E-02 = H( 62)
                      H( 3) =  -.21003040E-02 = H( 61)
                      H( 4) =  -.27069680E-02 = H( 60)
                      H( 5) =  -.33382180E-02 = H( 59)
                      H( 6) =  -.39567210E-02 = H( 58)
                      H( 7) =  -.45167690E-02 = H( 57)
                      H( 8) =  -.49655480E-02 = H( 56)
                      H( 9) =  -.52443800E-02 = H( 55)
                      H(10) =  -.52912720E-02 = H( 54)
                      H(11) =  -.50437710E-02 = H( 53)
                      H(12) =  -.44416870E-02 = H( 52)
                      H(13) =  -.34307100E-02 = H( 51)
                      H(14) =  -.19657400E-02 = H( 50)
                      H(15) =  -.13794770E-04 = H( 49)
                      H(16) =   .24416860E-02 = H( 48)
                      H(17) =   .54016280E-02 = H( 47)
                      H(18) =   .88445020E-02 = H( 46)
                      H(19) =   .12731530E-01 = H( 45)
                      H(20) =   .17000680E-01 = H( 44)
                      H(21) =   .21571820E-01 = H( 43)
                      H(22) =   .26347610E-01 = H( 42)
                      H(23) =   .31215140E-01 = H( 41)
                      H(24) =   .36051430E-01 = H( 40)
                      H(25) =   .40727130E-01 = H( 39)
                      H(26) =   .45110720E-01 = H( 38)
                      H(27) =   .49074520E-01 = H( 37)
                      H(28) =   .52500080E-01 = H( 36)
                      H(29) =   .55282880E-01 = H( 35)
                      H(30) =   .57335970E-01 = H( 34)
                      H(31) =   .58594040E-01 = H( 33)
                      H(32) =   .59018030E-01 = H( 32)

                              BAND  1          BAND  2
         LOWER BAND EDGE      .000000000       .050000000
         UPPER BAND EDGE      .010000000       .500000000
         DESIRED VALUE       1.000000000       .000000000
         WEIGHTING           1.000000000      1.000000000
         DEVIATION            .002450794       .002450794
         DEVIATION IN DB   -52.213810000    -52.213810000

         EXTREMA FREQUENCIES
            .0067708    .0100000    .0500000    .0536458    .0630208
            .0755208    .0890625    .1036458    .1187499    .1343751
            .1494794    .1651046    .1807298    .1968759    .2125011
            .2281263    .2442724    .2598973    .2760428    .2921884
            .3078132    .3239588    .3401044    .3557291    .3718747
            .3880203    .4041659    .4197907    .4359362    .4520818
            .4677066    .4838522    .5000000
```

Table C5 : Filter Coefficients for LPF1500 L9

```
*************************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

              FILTER LENGTH =   9

              ***** IMPULSE RESPONSE *****
                   H( 1) =  -.47385260E-01 = H(  9)
                   H( 2) =   .51096300E-01 = H(  8)
                   H( 3) =   .15307250E+00 = H(  7)
                   H( 4) =   .25925620E+00 = H(  6)
                   H( 5) =   .30436960E+00 = H(  5)

                           BAND  1          BAND  2
 LOWER BAND EDGE        .000000000      .200000000
 UPPER BAND EDGE        .100000000      .500000000
 DESIRED VALUE         1.000000000      .000000000
 WEIGHTING             1.000000000     1.300000000
 DEVIATION              .136449300      .104961000
 DEVIATION IN DB     -17.300560000   -19.579420000

 EXTREMA FREQUENCIES
     .0000000     .1000000     .2000000     .2633333     .3766665
     .5000000

*************************************************************
```

Table C6 : Filter Coefficients for LPF1500 L31

```
************************************************************

                         FINITE IMPULSE RESPONSE (FIR)
                         LINEAR PHASE DIGITAL FILTER DESIGN
                         REMEZ EXCHANGE ALGORITHM

                         BANDPASS FILTER

                  FILTER LENGTH =  31

                  ***** IMPULSE RESPONSE *****
                      H( 1) =   .18956240E-02 = H( 31)
                      H( 2) =   .16796450E-02 = H( 30)
                      H( 3) =  -.13467520E-02 = H( 29)
                      H( 4) =  -.62655380E-02 = H( 28)
                      H( 5) =  -.76840400E-02 = H( 27)
                      H( 6) =  -.33890070E-03 = H( 26)
                      H( 7) =   .13511050E-01 = H( 25)
                      H( 8) =   .21470150E-01 = H( 24)
                      H( 9) =   .94669810E-02 = H( 23)
                      H(10) =  -.22509910E-01 = H( 22)
                      H(11) =  -.51376640E-01 = H( 21)
                      H(12) =  -.41209470E-01 = H( 20)
                      H(13) =   .29889860E-01 = H( 19)
                      H(14) =   .14616060E+00 = H( 18)
                      H(15) =   .25564390E+00 = H( 17)
                      H(16) =   .30050370E+00 = H( 16)

                        BAND  1        BAND  2
LOWER BAND EDGE      .000000000     .200000000
UPPER BAND EDGE      .100000000     .500000000
DESIRED VALUE       1.000000000     .000000000
WEIGHTING          10.000000000   10.000000000
DEVIATION            .001523195     .001523195
DEVIATION IN DB   -56.344830000  -56.344830000

EXTREMA FREQUENCIES
     .0000000     .0437500     .0718750     .0927083     .1000000
     .2000000     .2072917     .2260416     .2520833     .2802087
     .3104174     .3406262     .3729182     .4041687     .4364608
     .4677112     .5000000

************************************************************
```

Table C7 : Filter Coefficients for LPF1500 L63

```
                         FINITE IMPULSE RESPONSE (FIR)
                         LINEAR PHASE DIGITAL FILTER DESIGN
                         REMEZ EXCHANGE ALGORITHM

                         BANDPASS FILTER

               FILTER LENGTH =  63

               ***** IMPULSE RESPONSE *****
                      H( 1) =  -.11497210E-04 = H( 63)
                      H( 2) =  -.40177820E-05 = H( 62)
                      H( 3) =   .40104340E-04 = H( 61)
                      H( 4) =   .94730820E-04 = H( 60)
                      H( 5) =   .61529720E-04 = H( 59)
                      H( 6) =  -.13574090E-03 = H( 58)
                      H( 7) =  -.38460420E-03 = H( 57)
                      H( 8) =  -.35381780E-03 = H( 56)
                      H( 9) =   .21458670E-03 = H( 55)
                      H(10) =   .10484500E-02 = H( 54)
                      H(11) =   .12620260E-02 = H( 53)
                      H(12) =   .64365110E-04 = H( 52)
                      H(13) =  -.21220690E-02 = H( 51)
                      H(14) =  -.33338030E-02 = H( 50)
                      H(15) =  -.14759420E-02 = H( 49)
                      H(16) =   .32193100E-02 = H( 48)
                      H(17) =   .70782760E-02 = H( 47)
                      H(18) =   .52989580E-02 = H( 46)
                      H(19) =  -.31939230E-02 = H( 45)
                      H(20) =  -.12623200E-01 = H( 44)
                      H(21) =  -.13302470E-01 = H( 43)
                      H(22) =  -.22721440E-03 = H( 42)
                      H(23) =   .19407710E-01 = H( 41)
                      H(24) =   .28124250E-01 = H( 40)
                      H(25) =   .11444060E-01 = H( 39)
                      H(26) =  -.26139810E-01 = H( 38)
                      H(27) =  -.56691010E-01 = H( 37)
                      H(28) =  -.43688850E-01 = H( 36)
                      H(29) =   .31144480E-01 = H( 35)
                      H(30) =   .14849420E+00 = H( 34)
                      H(31) =   .25652870E+00 = H( 33)
                      H(32) =   .30033450E+00 = H( 32)

                          BAND  1           BAND  2
LOWER BAND EDGE      .00000000         .20000000
UPPER BAND EDGE      .10000000         .50000000
DESIRED VALUE       1.00000000         .00000000
WEIGHTING          10.00000000       10.00000000
DEVIATION            .00008070         .00008070
DEVIATION IN DB -101.86220000 -101.86220000

EXTREMA FREQUENCIES
     .0000000     .0151042     .0302083     .0442708     .0583333
     .0708333     .0822916     .0916666     .0979166     .1000000
     .2000000     .2020834     .2083334     .2171877     .2286462
     .2411464     .2546881     .2687504     .2833335     .2979166
     .3130205     .3281245     .3437492     .3593740     .3744779
     .3901027     .4057274     .4213522     .4369770     .4526017
     .4687473     .4843721     .5000000
```

Table C8 : Filter Coefficients for BPF2500s L9

```
************************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

              FILTER LENGTH =    9

              ***** IMPULSE RESPONSE *****
                   H( 1) =    .20384600E+00 = H(  9)
                   H( 2) =   -.30669860E-07 = H(  8)
                   H( 3) =   -.14781930E+00 = H(  7)
                   H( 4) =   -.25204850E-09 = H(  6)
                   H( 5) =    .16120310E+00 = H(  5)

                          BAND  1         BAND  2         BAND  3
 LOWER BAND EDGE       .000000000      .230000000      .320000000
 UPPER BAND EDGE       .180000000      .270000000      .500000000
 DESIRED VALUE         .000000000     1.000000000      .000000000
 WEIGHTING            1.000000000     1.400000000     1.000000000
 DEVIATION             .273256500      .195183200      .273256500
 DEVIATION IN DB    -11.268580000   -14.191140000   -11.268580000

 EXTREMA FREQUENCIES
     .0000000     .1100000     .1800000     .2700000     .3200000
     .3899999

***********************************************************
```

Table C9 : Filter Coefficients for BPF2500s L15

```
************************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

              FILTER LENGTH =  15

              ***** IMPULSE RESPONSE *****
                   H( 1) =   .13267410E-04 = H( 15)
                   H( 2) =  -.60674130E-01 = H( 14)
                   H( 3) =   .31562800E-04 = H( 13)
                   H( 4) =   .13888770E+00 = H( 12)
                   H( 5) =   .33913420E-04 = H( 11)
                   H( 6) =  -.20118000E+00 = H( 10)
                   H( 7) =   .33453400E-04 = H(  9)
                   H( 8) =   .23848360E+00 = H(  8)

                             BAND  1         BAND  2         BAND  3
 LOWER BAND EDGE        .000000000      .230000000      .350000000
 UPPER BAND EDGE        .150000000      .270000000      .500000000
 DESIRED VALUE          .000000000     1.000000000      .000000000
 WEIGHTING            10.000000000    10.000000000    10.000000000
 DEVIATION              .039921910      .039921910      .039921910
 DEVIATION IN DB      -27.975740000   -27.975740000   -27.975740000

 EXTREMA FREQUENCIES
     .0585938    .1210938    .1500000    .2300000    .2495317
     .2700000    .3500000    .3773437    .4398437

************************************************************
```

Table C10 : Filter Coefficients for BPF2500s L31

```
*****************************************************

             ***** IMPULSE RESPONSE *****
                  H( 1) =    .87039240E-06 = H( 31)
                  H( 2) =    .90491730E-02 = H( 30)
                  H( 3) =   -.29628120E-05 = H( 29)
                  H( 4) =   -.18620890E-01 = H( 28)
                  H( 5) =   -.97091370E-06 = H( 27)
                  H( 6) =    .18941140E-01 = H( 26)
                  H( 7) =    .11878130E-05 = H( 25)
                  H( 8) =    .64614740E-02 = H( 24)
                  H( 9) =    .62477670E-06 = H( 23)
                  H(10) =   -.64797790E-01 = H( 22)
                  H(11) =    .23879300E-05 = H( 21)
                  H(12) =    .14371060E+00 = H( 20)
                  H(13) =   -.19549110E-05 = H( 19)
                  H(14) =   -.21299060E+00 = H( 18)
                  H(15) =    .82128430E-06 = H( 17)
                  H(16) =    .24064200E+00 = H( 16)

                            BAND  1           BAND  2           BAND  3
LOWER BAND EDGE          .000000000        .230000000        .350000000
UPPER BAND EDGE          .150000000        .270000000        .500000000
DESIRED VALUE            .000000000       1.000000000        .000000000
WEIGHTING              10.000000000      10.000000000      10.000000000
DEVIATION                .004148392        .004148392        .004148392
DEVIATION IN DB       -47.642350000     -47.642350000     -47.642350000

EXTREMA FREQUENCIES
      .0000000     .0312500     .0625000     .0937500     .1210938
      .1406250     .1500000     .2300000     .2592969     .2700000
      .3500000     .3578125     .3792969     .4066406     .4378906
      .4691406     .5000000

*****************************************************************
```

Table C11 : Filter Coefficients for BPF2500s L63

```
                    FINITE IMPULSE RESPONSE (FIR)
                    LINEAR PHASE DIGITAL FILTER DESIGN
                    REMEZ EXCHANGE ALGORITHM

                    BANDPASS FILTER

             FILTER LENGTH =  63

             ***** IMPULSE RESPONSE *****
                  H( 1) =  -.35531050E-06 = H( 63)
                  H( 2) =   .23697640E-03 = H( 62)
                  H( 3) =   .13197250E-05 = H( 61)
                  H( 4) =  -.61867540E-03 = H( 60)
                  H( 5) =  -.20625860E-05 = H( 59)
                  H( 6) =   .62498140E-03 = H( 58)
                  H( 7) =   .10481090E-05 = H( 57)
                  H( 8) =   .66400040E-03 = H( 56)
                  H( 9) =   .27838720E-05 = H( 55)
                  H(10) =  -.35488550E-02 = H( 54)
                  H(11) =  -.79394680E-05 = H( 53)
                  H(12) =   .63765240E-02 = H( 52)
                  H(13) =   .10006150E-04 = H( 51)
                  H(14) =  -.53608550E-02 = H( 50)
                  H(15) =  -.45091680E-05 = H( 49)
                  H(16) =  -.30799540E-02 = H( 48)
                  H(17) =  -.82562560E-05 = H( 47)
                  H(18) =   .18017490E-01 = H( 46)
                  H(19) =   .21076480E-04 = H( 45)
                  H(20) =  -.30820170E-01 = H( 44)
                  H(21) =  -.23591210E-04 = H( 43)
                  H(22) =   .26872950E-01 = H( 42)
                  H(23) =   .10135640E-04 = H( 41)
                  H(24) =   .66850530E-02 = H( 40)
                  H(25) =   .13716950E-04 = H( 39)
                  H(26) =  -.70965640E-01 = H( 38)
                  H(27) =  -.33312280E-04 = H( 37)
                  H(28) =   .15008030E+00 = H( 36)
                  H(29) =   .34467460E-04 = H( 35)
                  H(30) =  -.21587720E+00 = H( 34)
                  H(31) =  -.14526260E-04 = H( 33)
                  H(32) =   .24148710E+00 = H( 32)

                            BAND  1         BAND  2         BAND  3
LOWER BAND EDGE          .000000000      .230000000      .350000000
UPPER BAND EDGE          .150000000      .270000000      .500000000
DESIRED VALUE            .000000000     1.000000000      .000000000
WEIGHTING               1.000000000     1.000000000     1.000000000
DEVIATION                .000060967      .000060967      .000060967
DEVIATION IN DB       -84.297950000   -84.297950000   -84.297950000

EXTREMA FREQUENCIES
     .0000000    .0156250    .0302734    .0458984    .0615234
     .0761719    .0908203    .1044922    .1181641    .1298828
     .1406250    .1474609    .1500000    .2300000    .2329297
     .2397656    .2495313    .2602735    .2671094    .2700000
     .3500000    .3529297    .3597656    .3675781    .3822266
     .3949219    .4095703    .4242187    .4388672    .4544922
     .4691406    .4847656    .5000000
```

Table C12 : Filter Coefficients for BPF2500 L9

```
***********************************************************************

                                   FINITE IMPULSE RESPONSE (FIR)
                                   LINEAR PHASE DIGITAL FILTER DESIGN
                                   REMEZ EXCHANGE ALGORITHM

                                   BANDPASS FILTER

                          FILTER LENGTH =   9

                          ***** IMPULSE RESPONSE *****
                               H( 1) =   .59850840E-01 = H(  9)
                               H( 2) =   .83494020E-08 = H(  8)
                               H( 3) =  -.29925420E+00 = H(  7)
                               H( 4) =   .14653130E-07 = H(  6)
                               H( 5) =   .38029830E+00 = H(  5)

                                  BAND  1              BAND  2              BAND  3
 LOWER BAND EDGE          .000000000          .200000000          .400000000
 UPPER BAND EDGE          .100000000          .300000000          .500000000
 DESIRED VALUE            .000000000         1.000000000          .000000000
 WEIGHTING               1.000000000         1.000000000         1.000000000
 DEVIATION                .098508380          .098508380          .098508380
 DEVIATION IN DB       -20.130510000       -20.13051000c       -20.130510000

 EXTREMA FREQUENCIES
      .0000000       .1000000       .2000000       .2500000       .3000000
      .4000000

***********************************************************************
```

Table C13 : Filter Coefficients for BPF2500 L15

```
**********************************************************

                     FINITE IMPULSE RESPONSE (FIR)
                     LINEAR PHASE DIGITAL FILTER DESIGN
                     REMEZ EXCHANGE ALGORITHM

                     BANDPASS FILTER

           FILTER LENGTH =  15

           ***** IMPULSE RESPONSE *****
                H( 1) =   .57887090E-08 = H( 15)
                H( 2) =   .38867450E-01 = H( 14)
                H( 3) =  -.68377540E-08 = H( 13)
                H( 4) =   .71470100E-01 = H( 12)
                H( 5) =  -.14981210E-07 = H( 11)
                H( 6) =  -.28886740E+00 = H( 10)
                H( 7) =  -.43811290E-08 = H(  9)
                H( 8) =   .41073510E+00 = H(  8)

                           BAND  1         BAND  2         BAND  3
 LOWER BAND EDGE        .000000000      .200000000      .400000000
 UPPER BAND EDGE        .100000000      .300000000      .500000000
 DESIRED VALUE          .000000000     1.000000000      .000000000
 WEIGHTING             1.000000000     1.000000000     1.000000000
 DEVIATION              .053675240      .053675240      .053675240
 DEVIATION IN DB     -25.404490000   -25.404490000   -25.404490000

 EXTREMA FREQUENCIES
     .0000000    .0666667    .1000000    .2000000    .2500000
     .3000000    .4000000    .4333333    .5000000

**********************************************************
```

Table C14 : Filter Coefficients for BPF2500 L31

```
**********************************************************

                         FINITE IMPULSE RESPONSE (FIR)
                         LINEAR PHASE DIGITAL FILTER DESIGN
                         REMEZ EXCHANGE ALGORITHM

                         BANDPASS FILTER

                 FILTER LENGTH =  31

                 ***** IMPULSE RESPONSE *****
                      H( 1) =  -.42183190E-05 = H( 31)
                      H( 2) =  -.36066390E-02 = H( 30)
                      H( 3) =   .22711920E-05 = H( 29)
                      H( 4) =   .11444820E-01 = H( 28)
                      H( 5) =   .53756610E-05 = H( 27)
                      H( 6) =   .21266940E-02 = H( 26)
                      H( 7) =  -.66096860E-05 = H( 25)
                      H( 8) =  -.42699350E-01 = H( 24)
                      H( 9) =   .69611780E-06 = H( 23)
                      H(10) =   .42655130E-01 = H( 22)
                      H(11) =   .42237510E-05 = H( 21)
                      H(12) =   .84109580E-01 = H( 20)
                      H(13) =  -.29676600E-05 = H( 19)
                      H(14) =  -.29117750E+00 = H( 18)
                      H(15) =   .12313500E-05 = H( 17)
                      H(16) =   .39610960E+00 = H( 16)

                           BAND  1          BAND  2          BAND  3
 LOWER BAND EDGE        .000000000       .200000000       .400000000
 UPPER BAND EDGE        .100000000       .300000000       .500000000
 DESIRED VALUE          .000000000      1.000000000       .000000000
 WEIGHTING            10.000000000     10.000000000     10.000000000
 DEVIATION              .001815104       .001815104       .001815104
 DEVIATION IN DB      -54.821910000    -54.821910000    -54.821910000

 EXTREMA FREQUENCIES
     .0000000    .0390625    .0703125    .0917969    .1000000
     .2000000    .2078125    .2253906    .2507812    .2742187
     .2917969    .3000000    .4000000    .4078125    .4312500
     .4625000    .5000000

**********************************************************
```

Table C15 : Filter Coefficients for HPF2500 L31

```
*************************************************************

                         FINITE IMPULSE RESPONSE (FIR)
                         LINEAR PHASE DIGITAL FILTER DESIGN
                         REMEZ EXCHANGE ALGORITHM

                         BANDPASS FILTER

                FILTER LENGTH =  31

                ***** IMPULSE RESPONSE *****
                     H( 1) =    .21042800E-02 = H( 31)
                     H( 2) =   -.19346590E-05 = H( 30)
                     H( 3) =   -.46465720E-02 = H( 29)
                     H( 4) =    .73033790E-06 = H( 28)
                     H( 5) =    .94219370E-02 = H( 27)
                     H( 6) =   -.28609820E-05 = H( 26)
                     H( 7) =   -.17204930E-01 = H( 25)
                     H( 8) =    .20212450E-05 = H( 24)
                     H( 9) =    .29778450E-01 = H( 23)
                     H(10) =   -.51459510E-05 = H( 22)
                     H(11) =   -.51521800E-01 = H( 21)
                     H(12) =    .41184310E-05 = H( 20)
                     H(13) =    .98416370E-01 = H( 19)
                     H(14) =   -.47544330E-05 = H( 18)
                     H(15) =   -.31567870E+00 = H( 17)
                     H(16) =    .50000430E+00 = H( 16)

                           BAND  1         BAND  2
 LOWER BAND EDGE        .000000000      .300000000
 UPPER BAND EDGE        .200000000      .500000000
 DESIRED VALUE          .000000000     1.000000000
 WEIGHTING            10.000000000    10.000000000
 DEVIATION              .001349337      .001349337
 DEVIATION IN DB      -57.397530000   -57.397530000

 EXTREMA FREQUENCIES
     .0312500     .0625000     .0917969     .1210938     .1484375
     .1738281     .1933594     .2000000     .3000000     .3078125
     .3253906     .3507813     .3781250     .4074219     .4786719
     .4699219     .5000000

*************************************************************
```

Table C16 : Filter Coefficients for HPF3500 L9

```
**********************************************************

                        FINITE IMPULSE RESPONSE (FIR)
                        LINEAR PHASE DIGITAL FILTER DESIGN
                        REMEZ EXCHANGE ALGORITHM

                        BANDPASS FILTER

              FILTER LENGTH =    9

              ***** IMPULSE RESPONSE *****
                   H( 1) =  -.47385250E-01 = H(  9)
                   H( 2) =  -.51096310E-01 = H(  8)
                   H( 3) =   .15307250E+00 = H(  7)
                   H( 4) =  -.25925620E+00 = H(  6)
                   H( 5) =   .30436960E+00 = H(  5)

                        BAND  1        BAND  2
 LOWER BAND EDGE      .000000000     .400000000
 UPPER BAND EDGE      .300000000     .500000000
 DESIRED VALUE        .000000000    1.000000000
 WEIGHTING           1.300000000    1.000000000
 DEVIATION            .104961000     .136449300
 DEVIATION IN DB   -19.579420000  -17.300560000

 EXTREMA FREQUENCIES
     .0000000     .1233333     .2366665     .3000000     .4000000
     .5000000

**********************************************************
```

Table C17 : Filter Coefficients for HPF3500 L15

```
*****************************************************************

                    FINITE IMPULSE RESPONSE (FIR)
                    LINEAR PHASE DIGITAL FILTER DESIGN
                    REMEZ EXCHANGE ALGORITHM

                    BANDPASS FILTER

             FILTER LENGTH =  15

             ***** IMPULSE RESPONSE *****
                  H( 1) =   -.13284810E-01 = H( 15)
                  H( 2) =   -.22728370E-01 = H( 14)
                  H( 3) =    .44764240E-01 = H( 13)
                  H( 4) =   -.38067300E-01 = H( 12)
                  H( 5) =   -.27087680E-01 = H( 11)
                  H( 6) =    .14193710E+00 = H( 10)
                  H( 7) =   -.25439180E+00 = H(  9)
                  H( 8) =    .30128990E+00 = H(  8)

                            BAND  1         BAND  2
 LOWER BAND EDGE       .000000000       .400000000
 UPPER BAND EDGE       .300000000       .500000000
 DESIRED VALUE         .000000000      1.000000000
 WEIGHTING            1.000000000      1.000000000
 DEVIATION             .036427310       .036427310
 DEVIATION IN DB    -28.771430000    -28.771430000

 EXTREMA FREQUENCIES
      .0000000     .0703125     .1406250     .2109375     .2695313
      .3000000     .4000000     .4312500     .5000000

*****************************************************************
```

Table C18 : Filter Coefficients for HPF3500 L31

```
**********************************************************************

                          FINITE IMPULSE RESPONSE (FIR)
                          LINEAR PHASE DIGITAL FILTER DESIGN
                          REMEZ EXCHANGE ALGORITHM

                          BANDPASS FILTER

                 FILTER LENGTH =  31

                 ***** IMPULSE RESPONSE *****
                     H( 1) =  -.18968240E-02 = H( 31)
                     H( 2) =   .16820850E-02 = H( 30)
                     H( 3) =   .13447050E-02 = H( 29)
                     H( 4) =  -.62657850E-02 = H( 28)
                     H( 5) =   .76874290E-02 = H( 27)
                     H( 6) =  -.34516920E-03 = H( 26)
                     H( 7) =  -.13507420E-01 = H( 25)
                     H( 8) =   .21473700E-01 = H( 24)
                     H( 9) =  -.94763590E-02 = H( 23)
                     H(10) =  -.22502630E-01 = H( 22)
                     H(11) =   .51378460E-01 = H( 21)
                     H(12) =  -.41218850E-01 = H( 20)
                     H(13) =  -.29878980E-01 = H( 19)
                     H(14) =   .14615710E+00 = H( 18)
                     H(15) =  -.25565100E+00 = H( 17)
                     H(16) =   .30051690E+00 = H( 16)

                          BAND  1         BAND  2
 LOWER BAND EDGE       .000000000      .400000000
 UPPER BAND EDGE       .300000000      .500000000
 DESIRED VALUE         .000000000     1.000000000
 WEIGHTING           10.000000000    10.000000000
 DEVIATION             .001522161      .001522161
 DEVIATION IN DB    -56.350730000   -56.350730000

 EXTREMA FREQUENCIES
     .0000000     .0312500     .0644531     .0957031     .1269531
     .1582031     .1894531     .2207031     .2480469     .2734375
     .2929688     .3000000     .4000000     .4078125     .4273438
     .4566406     .5000000

**********************************************************************
```

Table C19 : Filter Coefficients for HPF3500 L63

```
                          FINITE IMPULSE RESPONSE (FIR)
                          LINEAR PHASE DIGITAL FILTER DESIGN
                          REMEZ EXCHANGE ALGORITHM

                          BANDPASS FILTER

                 FILTER LENGTH =  63

                 ***** IMPULSE RESPONSE *****
                     H( 1) =    .11472290E-04 = H( 63)
                     H( 2) =   -.39807700E-05 = H( 62)
                     H( 3) =   -.40098000E-04 = H( 61)
                     H( 4) =    .94604010E-04 = H( 60)
                     H( 5) =   -.61302900E-04 = H( 59)
                     H( 6) =   -.13588310E-03 = H( 58)
                     H( 7) =    .38436840E-03 = H( 57)
                     H( 8) =   -.35313870E-03 = H( 56)
                     H( 9) =   -.21528440E-03 = H( 55)
                     H(10) =    .10483880E-02 = H( 54)
                     H(11) =   -.12607540E-02 = H( 53)
                     H(12) =    .62518080E-04 = H( 52)
                     H(13) =    .21228730E-02 = H( 51)
                     H(14) =   -.33322140E-02 = H( 50)
                     H(15) =    .14724660E-02 = H( 49)
                     H(16) =    .32220220E-02 = H( 48)
                     H(17) =   -.70772930E-02 = H( 47)
                     H(18) =    .52939480E-02 = H( 46)
                     H(19) =    .31994880E-02 = H( 45)
                     H(20) =   -.12624310E-01 = H( 44)
                     H(21) =    .13296960E-01 = H( 43)
                     H(22) =   -.21857700E-03 = H( 42)
                     H(23) =   -.19412380E-01 = H( 41)
                     H(24) =    .28120110E-01 = H( 40)
                     H(25) =   -.11433360E-01 = H( 39)
                     H(26) =   -.26148700E-01 = H( 38)
                     H(27) =    .56690260E-01 = H( 37)
                     H(28) =   -.43679200E-01 = H( 36)
                     H(29) =   -.31156800E-01 = H( 35)
                     H(30) =    .14849800E+00 = H( 34)
                     H(31) =   -.25652060E+00 = H( 33)
                     H(32) =    .30032080E+00 = H( 32)

                            BAND  1         BAND  2
         LOWER BAND EDGE     .000000000      .400000000
         UPPER BAND EDGE     .300000000      .500000000
         DESIRED VALUE       .000000000     1.000000000
         WEIGHTING          1.000000000     1.000000000
         DEVIATION           .000008049      .000008049
         DEVIATION IN DB -101.884900000 -101.884900000

         EXTREMA FREQUENCIES
             .0000000     .0156250     .0312500     .0468750     .0625000
             .0781250     .0937500     .1093750     .1250000     .1406250
             .1562500     .1718750     .1865234     .2021484     .2167969
             .2314453     .2451172     .2587891     .2714844     .2822266
             .2919922     .2978516     .3000000     .4000000     .4019531
             .4087891     .4175781     .4292969     .4419922     .4556641
             .4703125     .4849609     .5000000
```

APPENDIX D

PROGRAM LISTINGS

```
(**********************************************************************)
(*                                                                    *)
(*                      program MAKEDATA_FILE                         *)
(*                                                                    *)
(*   This program allows the user to make a data file that contains   *)
(*   a signal with many different characteristics possible or some    *)
(*   combination of characteristics.  These possibilities are listed  *)
(*   in the first part of the main program and include various time   *)
(*   waveforms and two different noise distributions.  The program is *)
(*   designed to give the user maximum flexibility in selecting the   *)
(*   desired signal characteristics.                                  *)
(*                                                                    *)
(**********************************************************************)
program MAKEDATA_FILE(INPUT,OUTPUT);

const

   PI = 3.14159265358898;
   MAX_NUM_SAMPLES = 16384;

var

   CH      : CHAR;
   DATE,
   TIME    : ALFA;
   DATA_SAMPLES : TEXT;
   SEED1,
   SEED2,
   ANS,
   I,
   J              : INTEGER;
   A,
   U,
   R,
   O,
   X,
   DUMMY_NUM,
   INITIAL_VAL,
   SIGMA_SQ,
   LEVEL,
   PROBIMP,
   IMPULSE_VALUE,
   STEP_OFFSET,
   NOISE_AMPLITUDE : REAL;
   FILE_CHOICE,
   FREQ,
   CHOICE,
   START_FREQ,
   STOP_FREQ,
   START_SAMP,
   STOP_SAMP,
   NUMSAMP,
   NUMSTEPS,
   SIGN  : INTEGER;
   DONE : BOOLEAN;
   SIGNAL : array(.1..MAX_NUM_SAMPLES.) of REAL;

%INCLUDE CMS


(*--------------------------------------------------------------------*)
(*                                                                    *)
(*                          function SGN                              *)
(*                                                                    *)
(*   This function returns the sign of the TEST_VALUE.                *)
(*                                                                    *)
(*--------------------------------------------------------------------*)
function SGN ( TEST_VALUE : REAL ) : INTEGER;

begin

   SGN := 1;
   if TEST_VALUE < 0 then
      SGN := -1;

end;


begin                                          (*     of MAKEDATA_FILE      *)

   termin(INPUT);
   termout(OUTPUT);
   CMS('CLRSCRN',I);
   writeln('Would you like to 1) add to the existing signal');
   writeln('                  2) add to the ideal signal');
   writeln('             or else) make a new signal');
   readln(ANS);
   if (ANS = 1) or (ANS = 2) then
   begin
      case ANS of
         1 : reset(DATA_SAMPLES,'name=SIGNAL.PDATA.*');
         2 : reset(DATA_SAMPLES,'name=IDEAL.PDATA.*');
      end;
      readln(DATA_SAMPLES,I,DUMMY_NUM);
      while not(eof(DATA_SAMPLES)) do
         readln(DATA_SAMPLES,I,SIGNAL(.I.));
      NUMSAMP := I;
      writeln('There are',NUMSAMP:6,' samples of data');
   end
   else
   repeat
      writeln('How many samples to take (1024 recommended)?');
      readln(NUMSAMP);
      if (NUMSAMP > MAX_NUM_SAMPLES) or (NUMSAMP < 1) then
         writeln('The number of samples must be between 1 and 4096');
   until (NUMSAMP > 0) and (NUMSAMP <= MAX_NUM_SAMPLES);
   datetime(DATE,TIME);                         (*  generate random seed for *)
   readstr(str(TIME),I:2,CH,SEED1:2,CH,SEED2); (* random number           *)
   I := I*SEED1*SEED2+I+SEED1+SEED2;        (*      generator            *)
   U := random(I);
   writeln(I,U);
   writeln;
   writeln('BUILD A SIGNAL');
repeat
   writeln('What would you like in it?');
   writeln('            1) Constant level');
   writeln('            2) Monotone ramps');
   writeln('            3) Sinusoids');
   writeln('            4) Steps');
   writeln('            5) Gaussian noise');
   writeln('            6) Laplacian noise');
   writeln('            7) IIR filter the signal');
   writeln('            8) That''s all');
   readln(CHOICE);
   case CHOICE of

      1 : begin
   writeln('What shall the constant level be?');
   readln(LEVEL);
   if LEVEL <> 0 then
      for I := 1 to NUMSAMP do
         SIGNAL(.I.) := LEVEL;
          end;

      2 : begin
   writeln('What is the starting sample for the ramp?');
   readln(START_SAMP);
   writeln('And the ending sample?');
   readln(STOP_SAMP);
   writeln('To what relative value should the ramp rise(fall)?');
   readln(LEVEL);
   INITIAL_VAL := SIGNAL(.START_SAMP.);
   for I := START_SAMP to STOP_SAMP do
      SIGNAL(.I.) := INITIAL_VAL + LEVEL/abs(STOP_SAMP-START_SAMP)*
```

```
                                        (I-START_SAMP);
         end;

      3 : begin
   writeln('Input frequency of sinusiod in Hz(0 if group desired)');
   readln(FREQ);
   if FREQ <> 0 then
   begin
      writeln('starting sample(0 if all samples)?');
      readln(START_SAMP);
      if START_SAMP = 0 then
      begin
         START_SAMP := 1;
         STOP_SAMP := NUMSAMP;
      end
      else
      begin
         writeln('ending sample?');
         readln(STOP_SAMP);
      end;
      for I := START_SAMP to STOP_SAMP do
         SIGNAL(.I.) := SIGNAL(.I.) + sin(2*PI*FREQ*I/NUMSAMP);
   end
   else
   begin
      writeln('What is the starting frequency?');
      readln(START_FREQ);
      writeln('What is the last frequency?');
      readln(STOP_FREQ);
      for FREQ := START_FREQ to STOP_FREQ do
         for I := 1 to NUMSAMP do
            SIGNAL(.I.) := SIGNAL(.I.) + sin(2*PI*FREQ*I/NUMSAMP);
   end;
         end;

      4 : begin
   writeln('How many steps in the signal?');
   readln(NUMSTEPS);
   for J := 1 to NUMSTEPS do
      begin
         writeln('Input the starting sample for step',J);
         readln(START_SAMP);
         writeln('And the last sample?');
         readln(STOP_SAMP);
         writeln('And the offset?');
         readln(STEP_OFFSET);
         for I := START_SAMP to STOP_SAMP do
            SIGNAL(.I.) := SIGNAL(.I.) + STEP_OFFSET;
      end;
         end;

      5 : begin
   writeln('What shall the noise sigma squared be (gaussian dist.)?');
   writeln('Sigma squared?');
   readln(SIGMA_SQ);
   for I := 1 to NUMSAMP do
      begin
         U := random(0);
         R := sqrt(-2.0* SIGMA_SQ * ln(U));
         U := random(0);
         O := 2.0 * PI * U;
         X := R * cos(O);
         SIGNAL(.I.) := SIGNAL(.I.) + X;
      end;
         end;

      6 : begin
   writeln('What shall the probability of an impulse be?');
   readln(PROBIMP);
   if PROBIMP > 0 then
   begin
   writeln('Do you want 1) all positive impulses or 2) both negative',
          ' and positive impulses.');
   readln(SIGN);
   if SIGN = 2 then
      SIGN := -1;
   writeln('What shall the impulse standard deviation be?');
   readln(IMPULSE_VALUE);
   for J := 1 to NUMSAMP do
      begin
         U := random(0);
         if U <= PROBIMP then
         begin
            U := random(0);
            X := -(IMPULSE_VALUE/sqrt(2))*ln(1-U);
            SIGNAL(.J.) := SIGNAL(.J.) + X * SGN(random(0) +
                                (0.5 * SIGN));
         end;
      end;
   end;
          end;

      7 : begin
      writeln('What is the parameter ALPHA?');
      readln(A);
      for I := NUMSAMP downto 2 do
         SIGNAL(.I.) := A*SIGNAL(.I-1.) + SIGNAL(.I.);
          end;

      8 :
   DONE := true;

      OTHERWISE                          (*     do nothing            *)
          writeln('That is not an option');

   end;                                  (*     of case statement     *)

   CMS('CLRSCRN',I);

until DONE;


   writeln('Is this an 1) ideal signal or a 2) signal to be filtered?');
   readln(FILE_CHOICE);
   if FILE_CHOICE = 1 then
      rewrite(DATA_SAMPLES,'name=IDEAL.PDATA.*')
   else
      rewrite(DATA_SAMPLES,'name=SIGNAL.PDATA.*');
(* writeln(DATA_SAMPLES,0,NUMSAMP/1000.0:15:8); *)
   for I := 1 to NUMSAMP do
      writeln(DATA_SAMPLES,I:4,SIGNAL(.I.):15:8);

end;                                     (*     of MAKEDATA_FILE      *)
```

```
(******************************************************************)
(*                                                                *)
(*                         program FREQ                           *)
(*                                                                *)
(*   This program takes the discrete Fourier transform (DFT) of a *)
(*   signal. The signal may be up to 16384 elements long, but that *)
(*   length is not recommended due to the length of time to       *)
(*   calculate.  The DFT is implemented using a decimation-in-    *)
(*   frequency FFT algorithm.                                     *)
(*                                                                *)
(******************************************************************)
program FREQ (INPUT,OUTPUT);

%INCLUDE CMS

const

   PI = 3.1415926535898;
   MAX_ARRAY_SIZE = 16384;

var

   CHOICE,
   DUMMY,
   I,
   J,
   M,
   N,
   L,
   NV2,
   NM1,
   IP,
   LE,
   K      : INTEGER;
   LARGEST,
   TEST,
   LE1,
   UR,
   UI,
   WR,
   WI,
   TR,
   TI,
   TMR,
   TMI     : REAL;
   MAG,
   PHASE,
   XR,
   XI      : array(.1..MAX_ARRAY_SIZE.) of REAL;
   FILEA   : TEXT;


(*----------------------------------------------------------------*)
(*                                                                *)
(*                       function RAISE                           *)
(*                                                                *)
(*   This function raises A to an integer power N. This function is *)
(*   not intrinsically available in the Pascal compiler.          *)
(*                                                                *)
(*----------------------------------------------------------------*)
function RAISE ( A,N : INTEGER ) : INTEGER;

var

   TEMP,
   I : INTEGER;

begin

   TEMP := 1;
   for I := 1 to N do
      TEMP := TEMP * A;
   RAISE := TEMP;

end;


(*-------------------------------------------------------------------*)
(*                                                                   *)
(*                          function LOG                            *)
(*                                                                   *)
(*   This function implements the base 10 logarithm of X.  The Pascal *)
(*   compiler only gives an intrinsic natural logarithm.  The base 10 *)
(*   logarithm is calculated by                                      *)
(*                                                                   *)
(*                    log x = ln x                                   *)
(*                            -----                                  *)
(*                            ln 10                                  *)
(*                                                                   *)
(*-------------------------------------------------------------------*)
function LOG ( X : REAL ) : REAL;

begin

   if X = 0.0 then
      X := 0.00000001;
   LOG := ln(x)/ln(10.0);

end;



begin                                         (*      of FREQ              *)

   termin(INPUT);
   termout(OUTPUT);
   CMS('CLRSCRN',I);
   writeln('Of which file are we taking the DFT');
   writeln('            1) IDEAL');
   writeln('            2) SIGNAL');
   writeln('            3) GRAPH');
   writeln('            4) Rxx');
   readln(CHOICE);
   N := 0;
   if CHOICE <> 3 then
   begin
      case CHOICE of
         1 : reset(FILEA,'name=IDEAL.PDATA.*');
         2 : reset(FILEA,'name=SIGNAL.PDATA.*');
         4 : reset(FILEA,'name=Rxx.PDATA.*');
         otherwise
      end;
      if CHOICE <> 4 then
         readln(FILEA,DUMMY,XR( 1 ));     (*XR(1)->dummy read file size*)
      repeat                                  (* read in data to FFT      *)
         N := N + 1;
         readln(FILEA,DUMMY,XR( N ));
         XI( N ) := 0;
      until eof(FILEA);
   end
   else
   begin
      reset(FILEA,'name=GRAPH.PDATA.*');
      repeat
         N := N + 1;
         readln(FILEA,DUMMY,XI( N ),XR( N )); (* XI(N) is a dummy read*)
         XI( N ) := 0;
      until eof(FILEA);
   end;
   M := 0;                                    (* calculate M              *)
   TEST := N;
   repeat
      TEST := TEST/2;
      M := M + 1;
   until TEST <= 1;
   if TEST < 1 then
      for I := N+1 to RAISE(2,M) do      (* length /=2**M, pad with 0 *)
         XR(.I.) := 0.0;
   N := RAISE(2,M);
writeln(n);
   for L := 1 to M do                    (* calculate FFT             *)
   begin
      LE := RAISE(2,M+1-L);
      LE1 := LE / 2.0;
      UR := 1;
      UI := 0;
      WR := cos(PI/LE1);
      WI := -sin(PI/LE1);
      for J := 1 to round(LE1) do
      begin
         for K := round((J-1)/LE) to round(N/LE) do
         begin
            I := K*LE + J;
            if I <= N then
            begin
               IP := I + round(LE1);
               TR := XR(.I.) + XR(.IP.);
               TI := XI(.I.) + XI(.IP.);
               TMR :=XR(.I.) - XR(.IP.);
               TMI := XI(.I.) - XI(.IP.);
               XR(.IP.) := TMR*UR - TMI*UI;
               XI(.IP.) := TMR*UI + TMI*UR;
               XR(.I.) := TR;
               XI(.I.) := TI;
            end;
         end; (* NEXT K *)
         TR := UR*WR - UI*WI;
         UI := UR*WI + UI*WR;
         UR := TR;
      end; (* NEXT J *)
   end; (* NEXT L *)
   NV2 := N div 2;
   NM1 := N-1;
   J := 1;
   for I := 1 to NM1 do
   begin
      if I < J then
      begin
         TR := XR(.I.);
         XR(.I.) := XR(.J.);
         XR(.J.) := TR;
         TI := XI(.I.);
         XI(.I.) := XI(.J.);
         XI(.J.) := TI;
      end;
      K := NV2;
      while K < J do
      begin
         J := J-K;
         K := K div 2;
      end;
      J := J+K;
   end; (* NEXT I *)

   for I := 1 to N do                    (* calc mag & phase of DFT   *)
   begin
      MAG(.I.) := sqrt(sqr(XR(.I.)) + sqr(XI(.I.)));
      PHASE(.I.) := 0.0;
      if XR(.I.) <> 0.0 then
         PHASE(.I.) := arctan(XI(.I.)/XR(.I.)) * 180 / PI;
      if XR(.I.) < 0.0 then
         if XI(.I.) > 0.0 then
            PHASE(.I.) := 180 + PHASE(.I.)
         else
            PHASE(.I.) := -180 + PHASE(.I.)
   end;
   writeln('Would you like the output in LOG magnitude (1-yes,0-no)?');
   readln(CHOICE);
   if CHOICE = 1 then
      begin
         LARGEST := 0.0;
         for I := 1 to N do
            LARGEST := max(MAG(.I.),LARGEST);
         for I := 1 to N do
            MAG(.I.) := 20*log(MAG(.I.)/LARGEST);
      end;

   writeln('Is this 1) an X(w)');
   writeln('        2) a  Y(w)');
   writeln('  or else) a regular freq output');
   readln(CHOICE);
   case CHOICE of
      1 : rewrite(FILEA,'name=FREQX.PDATA.*');
      2 : rewrite(FILEA,'name=FREQY.PDATA.*');
      otherwise
         rewrite(FILEA,'name=FREQ.PDATA.*');
   end;
   for I := 1 to min(N,1024) do
      writeln(FILEA,I:4,MAG(.I.):15:8);

end.                                    (*          of FREQ          *)
```

```
(***********************************************************************)
(*                                                                     *)
(*                          program FILTER                             *)
(*                                                                     *)
(*     Author:  Doug Rider                                              *)
(*     Date  :  10 July 86                                             *)
(*                                                                     *)
(*     Purpose :  This program was designed to implement a generalized*)
(*              digital filter according to the following scheme.     *)
(*                                                                     *)
(*          Once a window size has been chosen the user is asked to    *)
(*          supply the requested weighting values or choose a          *)
(*          predefined set for either the time or rank weights.        *)
(*                                                                     *)
(*                                                                     *)
(*                                                                     *)
(*             time ordered  --------------------                      *)
(*               input -->  |    |    |  ...  |    |                   *)
(*                           --------------------'                     *)
(*                             ↑    ↑    ↑...     ↑                    *)
(*                      A(1)-O   -O   -O...A(n)-O                      *)
(*                             ↑    ↑    ↑        ↑                    *)
(*                           --------------------                      *)
(*                          |        SORT         |                    *)
(*                           --------------------                      *)
(*                             ↑    ↑    ↑...     ↑                    *)
(*                      B(1)-O   -O   -O...B(n)-O                      *)
(*                             ↑    ↑    ↑        ↑                    *)
(*                           --------------------                      *)
(*                          |   +    +   ..+..  +  | = output          *)
(*                           --------------------                      *)
(*                                                                     *)
(*               The program then opens  the external data             *)
(*               file where the values of the sampled                  *)
(*               signal to be filtered are and initializes the window. *)
(*               Initializing consists of filling the first half of the*)
(*               window with the first data value and then reading in  *)
(*               subsequent values until the window is filled.  Next   *)
(*               the window is weighted by the W() set of coefficients.*)
(*               The window is then rank ordered.  After this the      *)
(*               second set of coefficients are applied which trim off *)
(*               the smallest and largest values of the data.          *)
(*               After this last trimming the remaining samples are    *)
(*               summed and gain adjusted.  Finally, the               *)
(*               window is stepped over one value in time by deleting  *)
(*               the oldest value and inserting the new value in the   *)
(*               proper rank order.  The filtering and stepping        *)
(*               continue until the end of the data file is reached.   *)
(*                                                                     *)
(***********************************************************************)

program FILTER (INPUT,OUTPUT);

%INCLUDE CMS

const

   MAX_WINDOW_SIZE = 64;

type

   DATA_ELEMENT_POINTER = @DATA_ELEMENT;
   DATA_ELEMENT         = record
                               ACTUAL_VALUE : REAL;
                               MODIFIED_VAL : REAL;
                               NEXT_LARGEST : DATA_ELEMENT_POINTER;
                               NEXT_IN_TIME : DATA_ELEMENT_POINTER
                          end;
   WEIGHT_ARRAY_TYPE    = array(.1..MAX_WINDOW_SIZE.) of REAL;

var
```

```
   SMALLEST_VALUE,
   OLDEST_VALUE        : DATA_ELEMENT_POINTER;
   DATA_FILE_SIZE,
   I,
   WINDOW_SIZE         : INTEGER;
   DUMMY,
   CODED_FILE_SIZE,
   FILTER_INPUT,
   FILTER_OUTPUT       : REAL;
   GRAPH_FILE,
   DATA_SAMPLES        : TEXT;
   END_OF_DATA         : BOOLEAN;
   TIME_WEIGHT_ARRAY,
   RANK_WEIGHT_ARRAY   : WEIGHT_ARRAY_TYPE;

(*******************************************************************)
(*                                                                 *)
(*                     procedure GET_PARAMETERS                    *)
(*                                                                 *)
(*    This procedure returns the user input parameters that will be *)
(*    used during the program run.                                 *)
(*                                                                 *)
(*******************************************************************)

procedure GET_PARAMETERS ( var WINDOW_SIZE : INTEGER;
                           var TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                           var RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE);

var

   DC,
   ANS,
   CHOICE,
   I               : INTEGER;
   CHECK,
   VALID_SIZE      : BOOLEAN;
   GAIN,
   DUMMY           : REAL;
   COEFF_FILE      : TEXT;


(*-----------------------------------------------------------------*)
(*                                                                 *)
(*                     procedure MANUAL_INPUT                      *)
(*                                                                 *)
(*    This procedure lets the user input the weighting             *)
(*    coefficients to be used.                                     *)
(*                                                                 *)
(*-----------------------------------------------------------------*)

   procedure MANUAL_INPUT( var INPUT_ARRAY : WEIGHT_ARRAY_TYPE;
                               WINDOW_SIZE : INTEGER);

   var

      I,
      ANS     : INTEGER;

   begin

      writeln('Input the desired filter coefficients.');
      for I := 1 to round(WINDOW_SIZE/2) do
      begin
         writeln('A(',I:2,') = ?');
         readln(INPUT_ARRAY(.I.));
      end;
      writeln('Is the filter symmetric (1-yes,0-no)?');
      readln(ANS);
      for I := round(WINDOW_SIZE/2)+1 to WINDOW_SIZE do
         if ANS = 1 then
```

```
            INPUT_ARRAY(.I.) :=INPUT_ARRAY(.WINDOW_SIZE - I + 1.)
         else
         begin
            writeln('A(',I:2,') = ?');
            readln(INPUT_ARRAY(.I.));
         end;

   end;


(*--------------------------------------------------------------*)
(*                                                              *)
(*                   procedure DO_ALPHA_TRIM                    *)
(*                                                              *)
(*    This procedure calculates the number of elements to be    *)
(*    trimmed off of each end of the rank weight array and 0's them. *)
(*                                                              *)
(*--------------------------------------------------------------*)

   procedure DO_ALPHA_TRIM ( var RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                                 WINDOW_SIZE : INTEGER);

   var

      I,
      NUM_TRIMMED,
      TRIM,
      NUM_LEFT     : INTEGER;

   begin

      writeln('Input how many elements to trim (must be even #))');
      readln(NUM_TRIMMED);
      TRIM := NUM_TRIMMED div 2;
      NUM_LEFT := WINDOW_SIZE - NUM_TRIMMED;
      for I := 1 to TRIM do
         RANK_WEIGHT_ARRAY(.I.) := 0;
      for I := TRIM+1 to TRIM+NUM_LEFT do
         RANK_WEIGHT_ARRAY(.I.) := 1;
      for I := TRIM+NUM_LEFT+1 to WINDOW_SIZE do
         RANK_WEIGHT_ARRAY(.I.) := 0;

   end;


(*--------------------------------------------------------------*)
(*                                                              *)
(*                   procedure CALCULATE                        *)
(*                                                              *)
(*    This procedure calculates the gain factor for the filter  *)
(*    coefficients so that the output is properly scaled.       *)
(*                                                              *)
(*--------------------------------------------------------------*)

   procedure CALCULATE ( var GAIN : REAL;
                             WINDOW_SIZE : INTEGER;
                             TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                             RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE);

   var

      A,
      J,
      I    : INTEGER;
      ORDER : WEIGHT_ARRAY_TYPE;        (* this is the TWA ordered   *)

   begin

      ORDER(.1.) := TIME_WEIGHT_ARRAY(.1.);
      for I := 2 to WINDOW_SIZE do
         if TIME_WEIGHT_ARRAY(.I.) < ORDER(.1.) then
         begin
```

```
            for A := (I-1) downto 1 do
               ORDER(.A+1.) := ORDER(.A.);
            ORDER(.1.) := TIME_WEIGHT_ARRAY(.I.);
         end
         else
         begin
            J := 2;
            while (ORDER(.J.) <> 0) and (TIME_WEIGHT_ARRAY(.I.) >
                                          ORDER(.J.)) do
               J := J + 1;
            for A := (I-1) downto J do
               ORDER(.A+1.) := ORDER(.A.);
            ORDER(.J.) := TIME_WEIGHT_ARRAY(.I.);
         end;
      GAIN := 0.0;
      for I := 1 to WINDOW_SIZE do
         GAIN := GAIN + ORDER(.I.) * RANK_WEIGHT_ARRAY(.I.);

   end;


(*-----------------------------------------------------------------*)
(*                                                                 *)
(*                      procedure SHOW                             *)
(*                                                                 *)
(*    This procedure clears the screen and then provides a list    *)
(*    of the coefficients to the user.                             *)
(*                                                                 *)
(*-----------------------------------------------------------------*)

   procedure SHOW ( TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                    RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                    WINDOW_SIZE : INTEGER);

   var

      I  : INTEGER;

   begin
      CMS('CLRSCRN',I);
      writeln('TIME COEFF      VALUE       RANK COEFF      VALUE');
      for I := 1 to WINDOW_SIZE do
         writeln('  A(',I:2,') =',TIME_WEIGHT_ARRAY(.I.):12:5,
            '          B(',I:2,') =',RANK_WEIGHT_ARRAY(.I.):12:5);
   end;


(*-----------------------------------------------------------------*)
(*                                                                 *)
(*                      procedure MODIFY                           *)
(*                                                                 *)
(*    This procedure modifies either the whole weight array or     *)
(*    individual coefficient values that the user desires.         *)
(*                                                                 *)
(*-----------------------------------------------------------------*)

   procedure MODIFY ( var MODIFY_ARRAY : WEIGHT_ARRAY_TYPE;
                          WINDOW_SIZE : INTEGER);

   var

      I   : INTEGER;
      VAL : REAL;

   begin
      writeln('Which element ( 0-> all elements )');
      readln(I);
      if I = 0 then
        MANUAL_INPUT(MODIFY_ARRAY,WINDOW_SIZE)
      else
      begin
         writeln('What is the new value');
         readln(VAL);
         MODIFY_ARRAY(.I.) := VAL;
      end;
   end;



begin                                        (*       of GET_PARAMETERS  *)

   CMS('CLRSCRN',I);
   writeln;
   writeln('This program implements a digital filter.');
   writeln;
   repeat
      writeln('Input the desired window size(must be an odd integer).');
      writeln;
      read(WINDOW_SIZE);
      VALID_SIZE := true;
      if not(odd(WINDOW_SIZE)) then
         begin
            writeln('The window size must be an odd integer.');
            VALID_SIZE := false;
         end;
      if (WINDOW_SIZE > MAX_WINDOW_SIZE) or (WINDOW_SIZE < 0) then
         begin
            writeln('The size must be positive but less than ',
                     MAX_WINDOW_SIZE);
            VALID_SIZE := false;
         end;
   until VALID_SIZE;

   writeln('Would you like to initialize a predetermined filter');
   writeln('(i.e. a mean or median filter) or manually input the');
   writeln('filter coefficients?');
   writeln('                    1) mean filter');
   writeln('                    2) median filter');
   writeln('                    3) alpha-trimmed mean filter');
   writeln('                    4) alpha-trimmed linear filter');
   writeln('                    5) other coefficient input');
   readln(CHOICE);
   CMS('CLRSCRN',I);
   case CHOICE of
      1 : for I := 1 to WINDOW_SIZE do
          begin
             TIME_WEIGHT_ARRAY(.I.) := 1/WINDOW_SIZE;
             RANK_WEIGHT_ARRAY(.I.) := 1;
          end;
      2:  begin
             for I := 1 to WINDOW_SIZE do
             begin
                TIME_WEIGHT_ARRAY(.I.) := 1;
                RANK_WEIGHT_ARRAY(.I.) := 0;
             end;
             RANK_WEIGHT_ARRAY(.round(WINDOW_SIZE/2).) := 1;
          end;
      3:  begin

   for I := 1 to WINDOW_SIZE do
      TIME_WEIGHT_ARRAY(.I.) := 1;

   DO_ALPHA_TRIM(RANK_WEIGHT_ARRAY,WINDOW_SIZE);

   CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   for I := 1 to WINDOW_SIZE do
      RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;

          end;
      4:  begin

   CMS('CLRSCRN',I);
   writeln;
   writeln('Alpha-trimmed linear filter');
   writeln;
   writeln('Are the desired filter coefficients saved(1-yes,0-no)?');
   readln(ANS);
   if ANS = 1 then
   begin
      reset(COEFF_FILE,'name=FIR.COEFF.*');
      for I := 1 to WINDOW_SIZE do
         readln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
   end
   else
   begin
      CMS('CLRSCRN',I);
      MANUAL_INPUT(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
      writeln('Do you want to save this set (1-yes,0-no)?');
      readln(ANS);
      if ANS = 1 then
      begin
         rewrite(COEFF_FILE,'name=FIR.COEFF.*');
         for I := 1 to WINDOW_SIZE do
            writeln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
      end;
   end;

   CMS('CLRSCRN',I);
   DO_ALPHA_TRIM(RANK_WEIGHT_ARRAY,WINDOW_SIZE);

   CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   writeln('The gain for this set of coefficients is',GAIN:7:4);
   writeln;
   writeln('Is this a filter that passes dc (1-yes,0-no)?');
   readln(DC);
   if DC = 1 then
   begin
      writeln('The rank weighting coefficients have been divided');
      writeln(' by the gain in order to normalize the dc level');
      for I := 1 to WINDOW_SIZE do
         RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;
   end;

          end;
      5:  begin

   writeln('Are the desired filter coefficients saved(1-yes,0-no)?');
   readln(ANS);
   if ANS = 1 then
   begin
      reset(COEFF_FILE,'name=FIR.COEFF.*');
      for I := 1 to WINDOW_SIZE do
         readln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
   end
   else
   begin
      CMS('CLRSCRN',I);
      writeln;
   writeln('Now input the ',WINDOW_SIZE:2,' weight parameters for the');
      writeln('time ordered buffer');
      MANUAL_INPUT(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
      writeln('Do you want to save this set (1-yes,0-no)?');
      readln(ANS);
      if ANS = 1 then
      begin
         rewrite(COEFF_FILE,'name=FIR.COEFF.*');
         for I := 1 to WINDOW_SIZE do
            writeln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
      end;
   end;

   CMS('CLRSCRN',I);
   writeln;
   writeln('Now the same thing for the rank ordered buffer');
   MANUAL_INPUT(RANK_WEIGHT_ARRAY,WINDOW_SIZE);
   CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   for I := 1 to WINDOW_SIZE do
      RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;

          end;

   end;                                     (*      of case statement    *)

   writeln('Would you like to see the coefficients(1-yes,0-no)?');
   readln(I);
   if I = 1 then
   SHOW(TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY,WINDOW_SIZE);

   writeln('Would you like to change anything(1-yes,0-no)?');
   readln(ANS);
   while ANS = 1 do
   begin
      writeln('Which set of coefficients 1->time, 2->rank');
      readln(CHOICE);
      case CHOICE of
         1 : MODIFY(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
         2 : MODIFY(RANK_WEIGHT_ARRAY,WINDOW_SIZE);
      end;
      SHOW (TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY,WINDOW_SIZE);
      writeln('Change anything else(1-yes,0-no)?');
      readln(ANS);
   end;

end;                                        (*        of GET_PARAMETERS  *)



(********************************************************************)
(*                                                                  *)
(*                     procedure INITIALIZE                         *)
(*                                                                  *)
(*    This procedure initializes the buffer window by reading the   *)
(*    first data value and using it to fill the first half of the   *)
(*    window and then filling the rest of the window with the       *)
(*    subsequent data values read out of the data file.             *)
(*                                                                  *)
(********************************************************************)

procedure INITIALIZE (     WINDOW_SIZE : INTEGER;
                       var OLDEST_VALUE : DATA_ELEMENT_POINTER);
var

   ELEMENT,
   NEXT_ELEMENT  : DATA_ELEMENT_POINTER;
   DUMMY         : REAL;
   I,
   MIDDLE_OF_WINDOW : INTEGER;

begin                                       (*     of INITIALIZE         *)

   new(ELEMENT);                            (* allocate dynamic variable *)
   OLDEST_VALUE := ELEMENT;
   read(DATA_SAMPLES,DUMMY,ELEMENT@.ACTUAL_VALUE); (* read 1st value    *)
   MIDDLE_OF_WINDOW := round(WINDOW_SIZE/2); (* find middle of window *)

   for I := 2 to MIDDLE_OF_WINDOW do        (* fill 1st half of window   *)
      begin
         new(NEXT_ELEMENT);
         ELEMENT@.NEXT_IN_TIME := NEXT_ELEMENT; (* link data values   *)
         ELEMENT := NEXT_ELEMENT;
         ELEMENT@.ACTUAL_VALUE := OLDEST_VALUE@.ACTUAL_VALUE;
      end;

   for I := (MIDDLE_OF_WINDOW + 1) to WINDOW_SIZE do (* read in data  *)
      begin                                 (* for last half of window   *)
```

```
         new(NEXT_ELEMENT);
         read(DATA_SAMPLES,DUMMY,NEXT_ELEMENT@.ACTUAL_VALUE);
         ELEMENT@.NEXT_IN_TIME := NEXT_ELEMENT;
         ELEMENT := NEXT_ELEMENT;
      end;

end;                                        (*       of INITIALIZE       *)


(*******************************************************************)
(*                                                                 *)
(*                    procedure WEIGHT_TIME_BUFFER                 *)
(*                                                                 *)
(*    This procedure weights each of the elements in the time odered *)
(*    buffer by the user input values stored in the TIME_WEIGHT_ARRAY*)
(*                                                                 *)
(*******************************************************************)

procedure WEIGHT_TIME_BUFFER ( WINDOW_SIZE : INTEGER;
                               OLDEST_VALUE : DATA_ELEMENT_POINTER;
                               TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE );

var

   I : INTEGER;
   ELEMENT : DATA_ELEMENT_POINTER;

begin                                       (*  of WEIGHT_TIME_BUFFER    *)

   ELEMENT := OLDEST_VALUE;
   for I := 1 to WINDOW_SIZE do
      begin
         ELEMENT@.MODIFIED_VAL := ELEMENT@.ACTUAL_VALUE *
                                  TIME_WEIGHT_ARRAY(.I.);
         ELEMENT := ELEMENT@.NEXT_IN_TIME;
      end;

end;                                        (*  of WEIGHT_TIME_BUFFER    *)


(*******************************************************************)
(*                                                                 *)
(*                    procedure RANK_ORDER                         *)
(*                                                                 *)
(*    This procedure takes the window that has been modified by the  *)
(*    time weights & uses an insertion sort to rank order each of the*)
(*    elements.  At the end of this procedure each element in the    *)
(*    window is ordered by size of the value as well as in time.     *)
(*                                                                 *)
(*******************************************************************)

procedure RANK_ORDER (     WINDOW_SIZE : INTEGER;
                       var OLDEST_VALUE : DATA_ELEMENT_POINTER;
                       var SMALLEST_VALUE : DATA_ELEMENT_POINTER);

var

   ELEMENT,
   NEW_ELEMENT  : DATA_ELEMENT_POINTER;
   I            : INTEGER;

begin                                       (*       of RANK_ORDER       *)

   SMALLEST_VALUE := OLDEST_VALUE;          (* start with first value as *)
   SMALLEST_VALUE@.NEXT_LARGEST := nil;     (* smallest and insert each  *)
   NEW_ELEMENT := OLDEST_VALUE@.NEXT_IN_TIME; (* 1st new valu to insrt*)
   for I := 2 to WINDOW_SIZE do             (* loop through each value   *)
      begin                                 (* check if new valu<smallest*)
         ELEMENT := SMALLEST_VALUE;               (* value in order      *)
         if NEW_ELEMENT@.MODIFIED_VAL < SMALLEST_VALUE@.MODIFIED_VAL
            then begin
```

```
              NEW_ELEMENT@.NEXT_LARGEST := SMALLEST_VALUE;
              SMALLEST_VALUE := NEW_ELEMENT;
            end
        else                                   (* else find proper order    *)
            begin
              while (ELEMENT@.NEXT_LARGEST <> nil) and
                    (NEW_ELEMENT@.MODIFIED_VAL >
                     ELEMENT@.NEXT_LARGEST@.MODIFIED_VAL) do
                ELEMENT := ELEMENT@.NEXT_LARGEST;
                                               (* and insert in place       *)
              NEW_ELEMENT@.NEXT_LARGEST := ELEMENT@.NEXT_LARGEST;
              ELEMENT@.NEXT_LARGEST := NEW_ELEMENT;
            end;
        NEW_ELEMENT := NEW_ELEMENT@.NEXT_IN_TIME; (* then go on to   *)
      end;                                     (* insert next elmnt in order*)

end;                                           (*     of RANK_ORDER         *)

(********************************************************************)
(*                                                                  *)
(*                 procedure WEIGHT_RANK_BUFFER                     *)
(*                                                                  *)
(*    This procedure weights each of the elements in the time odered *)
(*    buffer by the user input values stored in the TIME_WEIGHT_ARRAY*)
(*                                                                  *)
(********************************************************************)

procedure WEIGHT_RANK_BUFFER ( WINDOW_SIZE : INTEGER;
                               SMALLEST_VALUE : DATA_ELEMENT_POINTER;
                               RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE );

var

   I : INTEGER;
   ELEMENT : DATA_ELEMENT_POINTER;

begin                                          (*  of RANK_WEIGHT_BUFFER    *)

   ELEMENT := SMALLEST_VALUE;
   for I := 1 to WINDOW_SIZE do
      begin
        ELEMENT@.MODIFIED_VAL := ELEMENT@.MODIFIED_VAL *
                                 RANK_WEIGHT_ARRAY(.I.);
        ELEMENT := ELEMENT@.NEXT_LARGEST;
      end;

end;                                           (*  of RANK_WEIGHT_BUFFER    *)


(********************************************************************)
(*                                                                  *)
(*                     procedure FILTER                             *)
(*                                                                  *)
(*    This procedure finds the middle value of the time sequence,   *)
(*    which is the input to the filter, and sums the values of the  *)
(*    order sequence, the filter output, and sends the two values   *)
(*    back to the main program to be stored in a graph file.        *)
(*                                                                  *)
(********************************************************************)

procedure FILTER      (     WINDOW_SIZE : INTEGER;
                        var OLDEST_VALUE : DATA_ELEMENT_POINTER;
                        var SMALLEST_VALUE : DATA_ELEMENT_POINTER;
                        var FILTER_INPUT : REAL;
                        var FILTER_OUTPUT : REAL);

var

   I,
   MIDDLE_OF_WINDOW : INTEGER;
   INPUT_ELEMENT,
   OUTPUT_ELEMENT   : DATA_ELEMENT_POINTER;

begin                                      (*       of FILTER          *)

   MIDDLE_OF_WINDOW := round(WINDOW_SIZE/2);
   INPUT_ELEMENT    := OLDEST_VALUE;
   OUTPUT_ELEMENT   := SMALLEST_VALUE;
   FILTER_OUTPUT := 0;
   for I := 1 to WINDOW_SIZE do            (* step thru window to calc  *)
      begin                                (* output                    *)
         if I < MIDDLE_OF_WINDOW then      (* filter input is middle val*)
            INPUT_ELEMENT := INPUT_ELEMENT@.NEXT_IN_TIME;
         FILTER_OUTPUT := FILTER_OUTPUT + OUTPUT_ELEMENT@.MODIFIED_VAL;
         OUTPUT_ELEMENT := OUTPUT_ELEMENT@.NEXT_LARGEST;
      end;
   FILTER_INPUT := INPUT_ELEMENT@.ACTUAL_VALUE;

end;                                       (*       of FILTER          *)


(*********************************************************************)
(*                                                                   *)
(*                        procedure STEP_ONE                         *)
(*                                                                   *)
(*     Step the window over one data value by deleting out the oldest *)
(*     value from the time sequence and then creating                *)
(*     a new dynamic variable, reading in its new value, and         *)
(*     inserting it at the end of the time sequence.                 *)
(*                                                                   *)
(*********************************************************************)

procedure STEP_ONE   (var OLDEST_VALUE : DATA_ELEMENT_POINTER;
                          END_OF_DATA : BOOLEAN);

var

   DUMMY : REAL;
   ELEMENT,
   TARGET_ELEMENT : DATA_ELEMENT_POINTER;

begin                                      (*        of STEP_ONE       *)

   (* * * *  delete oldest value from window  * * * *)

   TARGET_ELEMENT := OLDEST_VALUE;
   OLDEST_VALUE := OLDEST_VALUE@.NEXT_IN_TIME; (* delete from time seq*)

   dispose(TARGET_ELEMENT);                (* de-allocate dynamic varbl *)

   (* * * *  put new element into window  * * * *)

   new(ELEMENT);
   TARGET_ELEMENT := OLDEST_VALUE;         (* start at beginning to find*)
   repeat                                  (* the most recent value     *)
      TARGET_ELEMENT := TARGET_ELEMENT@.NEXT_IN_TIME;
   until TARGET_ELEMENT@.NEXT_IN_TIME = nil;
   TARGET_ELEMENT@.NEXT_IN_TIME := ELEMENT;          (* add new value *)
   ELEMENT@.NEXT_IN_TIME := nil;           (* and put nil value on end  *)
   if not(END_OF_DATA) then                (* if still data values left *)
      read(DATA_SAMPLES,DUMMY,ELEMENT@.ACTUAL_VALUE) (* read new value*)
   else                                    (* else put last data val in *)
      ELEMENT@.ACTUAL_VALUE := TARGET_ELEMENT@.ACTUAL_VALUE; (* buffer*)

end;                                       (*        of STEP_ONE       *)


(*********************************************************************)
```

```
(*                                                                    *)
(*                          MAIN PROGRAM                              *)
(*                                                                    *)
(*   The main program first initializes the terminal for input and    *)
(*   output and then gets the necessary parameters to run the program *)
(*   from the user. The data file is opened and filtered and the      *)
(*   results are stored in a file to be graphed later                 *)
(*                                                                    *)
(**********************************************************************)

begin

   termin(INPUT);                        (* open terminal for input     *)
   termout(OUTPUT);                      (* open terminal for output    *)
   GET_PARAMETERS(WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   reset(DATA_SAMPLES,'name=SIGNAL.PDATA.*');
   readln(DATA_SAMPLES,DUMMY,CODED_FILE_SIZE);
   DATA_FILE_SIZE := round(1000 * CODED_FILE_SIZE);
   rewrite(GRAPH_FILE,'name=GRAPH.PDATA.*');
   INITIALIZE(WINDOW_SIZE,OLDEST_VALUE);
   END_OF_DATA := FALSE;
   for I := 1 to DATA_FILE_SIZE do
   begin
      WEIGHT_TIME_BUFFER(WINDOW_SIZE,OLDEST_VALUE,TIME_WEIGHT_ARRAY);
      RANK_ORDER(WINDOW_SIZE,OLDEST_VALUE,SMALLEST_VALUE);
      WEIGHT_RANK_BUFFER(WINDOW_SIZE,SMALLEST_VALUE,RANK_WEIGHT_ARRAY);
      FILTER(WINDOW_SIZE,OLDEST_VALUE,SMALLEST_VALUE,FILTER_INPUT,
             FILTER_OUTPUT);
      writeln(GRAPH_FILE,I,' ',FILTER_INPUT,' ',FILTER_OUTPUT);
      if I >= (DATA_FILE_SIZE - round(WINDOW_SIZE/2) + 1) then
         END_OF_DATA := TRUE;
      STEP_ONE(OLDEST_VALUE,END_OF_DATA);
   end;

end.
```

```
(***************************************************************)
(*                                                             *)
(*                       program NMSE                          *)
(*                                                             *)
(*   This program calculates the normalized mean square error (NMSE)   *)
(*   and normalized average error (NAE) between two data files.  Care *)
(*   must be taken to insure which file is the ideal file to be used  *)
(*   as the reference for normalization.  This program also generates *)
(*   two files to graph later.  One file contains both signals that   *)
(*   the error was calculated between and the other file contains     *)
(*   the difference between the two signals.                          *)
(*                                                             *)
(***************************************************************)
program NMSE (INPUT,OUTPUT);

%INCLUDE CMS

var

   I,
   DUMMY,
   CHOICE,
   ANS,
   NUM_SAMPLES   : INTEGER;
   DUMMY_2,
   IDEAL_VALUE,
   ACTUAL_VALUE,
   DIFF,
   NMSE_NORM,
   NAE_NORM,
   NAE,
   NMSE            : REAL;
   FILEA,
   FILEC,
   FILED,
   FILEB         : TEXT;

begin

   termin(INPUT);
   termout(OUTPUT);
   CMS('CLRSCRN',I);
   writeln('Calculate the NMSE and NAE between the');
   writeln('                 1) ideal signal and noise corrupted signal');
   writeln('                 2) ideal signal and filter output');
   writeln('                 3) filter input and output');
   writeln('                 4) ideal and modified frequency response');
   writeln('                 5) X(w) and Y(w)');
   readln(CHOICE);
   case CHOICE of
      1:  begin
             reset(FILEA,'name=IDEAL.PDATA.*');
             reset(FILEB,'name=SIGNAL.PDATA.*');
          end;
      2:  begin
             reset(FILEA,'name=IDEAL.PDATA.*');
             reset(FILEB,'name=GRAPH.PDATA.*');
          end;
      3:  begin
             reset(FILEA,'name=SIGNAL.PDATA.*');
             reset(FILEB,'name=GRAPH.PDATA.*');
          end;
      4:  begin
             reset(FILEA,'name=IDEALRES.PDATA.*');
             reset(FILEB,'name=FREQ.PDATA.*');
          end;
      5:  begin
             reset(FILEA,'name=FREQX.PDATA.*');
             reset(FILEB,'name=FREQY.PDATA.*');
          end;
      otherwise
          writeln('That is not an option');
```

```
end;
NMSE := 0;
NMSE_NORM := 0;
NAE := 0;
NAE_NORM := 0;
if CHOICE < 4 then
   begin
      readln(FILEA,DUMMY,DUMMY_2);
      NUM_SAMPLES := round(1000 * DUMMY_2);
   end;
if CHOICE = 1 then
   readln(FILEB,DUMMY,DUMMY_2);
         rewrite(FILEC,'name=CHANGE.PDATA.*');
         rewrite(FILED,'name=DUALRESP.PDATA.*');
case CHOICE of
   1:  for I := 1 to NUM_SAMPLES do
       begin
          readln(FILEA,DUMMY,IDEAL_VALUE);
          readln(FILEB,DUMMY,ACTUAL_VALUE);
          DIFF := IDEAL_VALUE - ACTUAL_VALUE;
          NMSE := NMSE + sqr(DIFF);
          NMSE_NORM := NMSE_NORM + sqr(IDEAL_VALUE);
          NAE := NAE + abs(DIFF);
          NAE_NORM := NAE_NORM + abs(IDEAL_VALUE);
          writeln(FILED,DUMMY:4,IDEAL_VALUE:15:8,ACTUAL_VALUE:15:8);
             writeln(FILEC,DUMMY:4,DIFF:15:8);
       end;
   2,(* 2 or 3 *)
   3:  for I := 1 to NUM_SAMPLES do
       begin
          readln(FILEA,DUMMY,IDEAL_VALUE);
          readln(FILEB,DUMMY,DUMMY_2,ACTUAL_VALUE);
          writeln(FILED,DUMMY:4,IDEAL_VALUE:15:8,ACTUAL_VALUE:15:8);
          DIFF := IDEAL_VALUE - ACTUAL_VALUE;
             writeln(FILEC,DUMMY:4,DIFF:15:8);
          NMSE := NMSE + sqr(DIFF);
          NMSE_NORM := NMSE_NORM + sqr(IDEAL_VALUE);
          NAE := NAE + abs(DIFF);
          NAE_NORM := NAE_NORM + abs(IDEAL_VALUE);
       end;
   4,(* 4 or 5 *)
   5:  begin
          writeln('Are the frequency files in dB(1-yes,0-no');
          readln(ANS);
          while not(eof(FILEA)) do
          begin
             readln(FILEA,DUMMY,IDEAL_VALUE);
             readln(FILEB,DUMMY,ACTUAL_VALUE);
          writeln(FILED,DUMMY:4,IDEAL_VALUE:15:8,ACTUAL_VALUE:15:8);
             if ANS = 1 then
             begin
                IDEAL_VALUE := exp(IDEAL_VALUE*ln(10)/20.0);
                ACTUAL_VALUE := exp(ACTUAL_VALUE*ln(10)/20.0);
             end;
             DIFF := IDEAL_VALUE - ACTUAL_VALUE;
             NMSE := NMSE + sqr(DIFF);
             NMSE_NORM := NMSE_NORM + sqr(IDEAL_VALUE);
             NAE := NAE + abs(DIFF);
             NAE_NORM := NAE_NORM + abs(IDEAL_VALUE);
             writeln(FILEC,DUMMY:4,DIFF:15:8);
          end;
       end;
end;
NMSE := NMSE / NMSE_NORM;
NAE := NAE / NAE_NORM;
writeln('The normalized MSE is',NMSE:15:8);
writeln('The normalized average error is',NAE:15:8);

end.
```

```
(****************************************************************)
(****************************************************************)
(*                                                              *)
(*                    program MEGAMEDIAN                        *)
(*                                                              *)
(*    This program implements a series of procedures which will *)
(*  create a set of data, filter it and then generate the transfer *)
(*  function for the filter.  The program will loop through several *)
(*  runs and average the results to get a better approximation  *)
(*  to the transfer function if a random input to the filter is *)
(*  used.                                                       *)
(*                                                              *)
(*    The program has the following filter types already        *)
(*  implemented:                                                *)
(*        - mean filter                                         *)
(*        - median filter                                       *)
(*        - alpha-trimmed mean filter (ATMF)                    *)
(*  and the new filter we are trying to characterize: the       *)
(*        - alpha-trimmed linear filter (ATLF)                  *)
(*                                                              *)
(*                                                              *)
(*  PROGRAM VARIABLES :                                         *)
(*                                                              *)
(*     SIGNAL : This array holds the values of the signal to be *)
(*              filtered.                                       *)
(*                                                              *)
(*     FFTX_MAG,                                                *)
(*     FFTY_MAG : The magnitudes of the FFT of the input signal (X) *)
(*                and the filter output (Y).                    *)
(*                                                              *)
(*     Tyx_MAG : The transfer function of the filter obtained   *)
(*               using the cross power spectrum = Pxy/Pxx.      *)
(*                                                              *)
(*     Tyx_SUM : Array used to sum up the above variable in order *)
(*               to average the result  over a number of random *)
(*               inputs.                                        *)
(*                                                              *)
(*     SIGNAL_SIZE : The number of samples taken for each signal. *)
(*                                                              *)
(*     TRIALS : The number of random trials to average.         *)
(*                                                              *)
(*     I,                                                       *)
(*     J : Index variables.                                     *)
(*                                                              *)
(*                                                              *)
(*  In this program the main procedures are set off by a single line *)
(*  of asterisks. Sub-procedures are set off by a single line of *)
(*  dashes. Functions or other small procedures are set off by a *)
(*  line of alternating dashes and spaces.                      *)
(*                                                              *)
(*         main procedures -> *********************             *)
(*                                                              *)
(*         sub-proceures   -> ---------------------             *)
(*                                                              *)
(*         functions or other                                   *)
(*         small procedures-> - - - - - - - - - - -             *)
(*                                                              *)
(****************************************************************)
(****************************************************************)
program MEGAMEDIAN (INPUT,OUTPUT);

   %INCLUDE CMS

type

   SIGNAL_TYPE = array(.-31..1056.) of REAL;
   MEDIAN_TYPE = array(.1..128,1..500.) of REAL;

var

   SIGNAL,
   FFTX_MAG,
   FFTY_MAG,
   Tyx_MAG    : SIGNAL_TYPE;
   Tyx_SUM    : MEDIAN_TYPE;
   SIGNAL_SIZE,
   TRIALS,
   SEED1,
   SEED2,
   J,
   I          : INTEGER;
   DUMMY      : REAL;
   DATE,
   TIME       : ALFA;
   CH         : CHAR;

(******************************************************************)
(*                                                                *)
(*                       procedure SORT                           *)
(*                                                                *)
(*  This procedure is used to sort an array of SIZE elements into *)
(*  order.  It is found here at the beginning of the program since*)
(*  it is used in both the CALCULATE procedure (in FILTER) to sort*)
(*  the elements of the TIME_WEIGHT_ARRAY into order so that the  *)
(*  gain factor may be calculated and in the MEDIAN_AND_PRINT     *)
(*  procedure to sort the 500 elements of the transfer function at*)
(*  each of 256 points.                                           *)
(*                                                                *)
(******************************************************************)
procedure SORT (     SIZE : INTEGER;
                 var SORT_ARRAY : SIGNAL_TYPE );

var

   A,
   I,
   J     : INTEGER;
   ORDER : SIGNAL_TYPE;

begin

      for I := 1 to SIZE do
         ORDER(.I.) := 0.0;
      ORDER(.1.) := SORT_ARRAY(.1.);
      for I := 2 to SIZE do
         if SORT_ARRAY(.I.) < ORDER(.1.) then
         begin
            for A := (I-1) downto 1 do
               ORDER(.A+1.) := ORDER(.A.);
            ORDER(.1.) := SORT_ARRAY(.I.);
         end
         else
         begin
            J := 2;
            while (ORDER(.J.) <> 0) and (SORT_ARRAY(.I.) >
                                         ORDER(.J.)) do
               J := J + 1;
            for A := (I-1) downto J do
               ORDER(.A+1.) := ORDER(.A.);
            ORDER(.J.) := SORT_ARRAY(.I.);
         end;
      for I := 1 to SIZE do
         SORT_ARRAY(.I.) := ORDER(.I.);

end;

(******************************************************************)
(*                                                                *)
(*                     procedure MAKEDATA                         *)
(*                                                                *)
(*  This procedure generates the input sequences to be filtered   *)
(*  Many different signal characteristics are possible in this    *)
(*   procedure.  They are outlined in the first part of the main body *)
(*   of this procedure.                                                *)
(*                                                                     *)
(**********************************************************************)
procedure MAKEDATA ( var SIGNAL : SIGNAL_TYPE );

const

   PI = 3.14159265358980;
   MAX_NUM_SAMPLES = 1024;

var

   A,
   U,
   R,
   D,
   X,
   DUMMY_NUM,
   INITIAL_VAL,
   SIGMA_SQ,
   LEVEL,
   PROBIMP,
   IMPULSE_VALUE,
   STEP_OFFSET,
   NOISE_AMPLITUDE : REAL;
   ANS,
   I,
   J,
   FILE_CHOICE,
   FREQ,
   CHOICE,
   START_FREQ,
   STOP_FREQ,
   START_SAMP,
   STOP_SAMP,
   NUMSAMP,
   NUMSTEPS,
   SIGN : INTEGER;
   DONE : BOOLEAN;
   DATA_SAMPLES : TEXT;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
(*                                                                     *)
(*                          function SGN                               *)
(*                                                                     *)
(*    This function returns the sign of the TEST_VALUE.  This          *)
(*    is not available as an intrinsic Pascal function.                *)
(*                                                                     *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
function SGN ( TEST_VALUE : REAL ) : INTEGER;

begin

   SGN := 1;
   if TEST_VALUE < 0 then
      SGN := -1;

end;


begin                                       (*    of MAKEDATA          *)

   CMS('CLRSCRN',I);
   repeat
      writeln('How many samples to take (1024 recommended)?');
      readln(NUMSAMP);
      if (NUMSAMP > MAX_NUM_SAMPLES) or (NUMSAMP < 1) then
         writeln('The number of samples must be between 1 and 4096');
   until (NUMSAMP > 0) and (NUMSAMP <= MAX_NUM_SAMPLES);
   SIGNAL_SIZE := NUMSAMP;
```

```
    for I := -31 to 1056 do
       SIGNAL(.I.) := 0;
    writeln;
    writeln('BUILD A SIGNAL');
repeat
    writeln('What would you like in it?');
    writeln('           1) Constant level');
    writeln('           2) Monotone ramps');
    writeln('           3) Sinusoids');
    writeln('           4) Steps');
    writeln('           5) Gaussian noise');
    writeln('           6) Laplacian noise');
    writeln('           7) IIR filter the signal');
    writeln('           8) That''s all');
    readln(CHOICE);
    case CHOICE of

       1 : begin                         (*   constant level          *)
    writeln('What shall the constant level be?');
    readln(LEVEL);
    if LEVEL <> 0 then
       for I := 1 to NUMSAMP do
          SIGNAL(.I.) := LEVEL;
           end;

       2 : begin                         (*           ramp            *)
    writeln('What is the starting sample for the ramp?');
    readln(START_SAMP);
    writeln('And the ending sample?');
    readln(STOP_SAMP);
    writeln('To what relative value should the ramp rise(fall)?');
    readln(LEVEL);
    INITIAL_VAL := SIGNAL(.START_SAMP.);
    for I := START_SAMP to STOP_SAMP do
       SIGNAL(.I.) := INITIAL_VAL + LEVEL/abs(STOP_SAMP-START_SAMP)*
                                       (I-START_SAMP);
           end;

       3 : begin                         (* sines and blocks of sines *)
    writeln('Input frequency of sinusiod in Hz(0 if group desired)');
    readln(FREQ);
    if FREQ <> 0 then
    begin
       writeln('starting sample(0 if all samples)?');
       readln(START_SAMP);
       if START_SAMP = 0 then
       begin
          START_SAMP := 1;
          STOP_SAMP := NUMSAMP;
       end
       else
       begin
          writeln('ending sample?');
          readln(STOP_SAMP);
       end;
       for I := START_SAMP to STOP_SAMP do
          SIGNAL(.I.) := SIGNAL(.I.) + sin(2*PI*FREQ*I/NUMSAMP);
    end
    else
    begin
       writeln('What is the starting frequency?');
       readln(START_FREQ);
       writeln('What is the last frequency?');
       readln(STOP_FREQ);
       for FREQ := START_FREQ to STOP_FREQ do
          for I := 1 to NUMSAMP do
             SIGNAL(.I.) := SIGNAL(.I.) + sin(2*PI*FREQ*I/NUMSAMP);
    end;
           end;

       4 : begin                         (*    steps and impulses     *)
    writeln('How many steps in the signal?');
```

```
readln(NUMSTEPS);
for J := 1 to NUMSTEPS do
   begin
      writeln('Input the starting sample for step',J);
      readln(START_SAMP);
      writeln('And the last sample?');
      readln(STOP_SAMP);
      writeln('And the offset?');
      readln(STEP_OFFSET);
      for I := START_SAMP to STOP_SAMP do
         SIGNAL(.I.) := SIGNAL(.I.) + STEP_OFFSET;
   end;
       end;

   5 : begin                          (*         Gaussian noise    *)
writeln('What shall the noise sigma squared be (gaussian dist.)?');
writeln('Sigma squared?');
readln(SIGMA_SQ);
for I := 1 to NUMSAMP do
   begin
      U := random(0);
      R := sqrt(-2.0* SIGMA_SQ * ln(u));
      U := random(0);
      O := 2.0 * PI * U;
      X := R * cos(O);
      SIGNAL(.I.) := SIGNAL(.I.) + X;
   end;
       end;

   6 : begin                          (* Laplacian(impulsive) noise*)
writeln('What shall the probability of an impulse be?');
readln(PROBIMP);
if PROBIMP > 0 then
begin
writeln('Do you want 1) all positive impulses or 2) both negative',
        ' and positive impulses.');
readln(SIGN);
if SIGN = 2 then
   SIGN := -1;
writeln('What shall the impulse standard deviation be?');
readln(IMPULSE_VALUE);
for J := 1 to NUMSAMP do
   begin
      U := random(0);
      if U <= PROBIMP then
      begin
         U := random(0);
         X := -(IMPULSE_VALUE/sqrt(2))*ln(1-U);
         SIGNAL(.J.) := SIGNAL(.J.) + X * SGN(random(0) +
                        (0.5 * SIGN));
      end;
   end;
end;
       end;

   7 : begin                          (* simple IIR low pass filter*)
   writeln('What is the parameter ALPHA?');
   readln(A);
   for I := NUMSAMP downto 2 do
      SIGNAL(.I.) := A*SIGNAL(.I-1.) + SIGNAL(.I.);
       end;

   8 :
DONE := true;

   OTHERWISE                        (*      do nothing            *)
       writeln('That is not an option');

end;                                (*      of case statement     *)

CMS('CLRSCRN',I);
```

```
until DONE;

end;                                        (*       of MAKEDATA          *)


(*********************************************************************)
(*                                                                   *)
(*                       procedure WINDOW                            *)
(*                                                                   *)
(*   This procedure windows the input data to the filter with the    *)
(*   Blackman window.  This window is given by                       *)
(*                                                                   *)
(*          w(n) = 0.42 - 0.5cos(2nPi/N-1) + 0.08cos(4nPi/N-1)       *)
(*                                                                   *)
(*   where N-1 is the signal size.                                   *)
(*                                                                   *)
(*********************************************************************)
procedure WINDOW ( var SIGNAL : SIGNAL_TYPE );

const

   PI = 3.14159265358898;

var

   MODIFIER : REAL;
   I : INTEGER;

begin

   for I := 1 to SIGNAL_SIZE do
   begin
      MODIFIER := 0.42-0.5*cos(2*PI*I/SIGNAL_SIZE)+0.08*cos(4*PI*I/
                                                       SIGNAL_SIZE);
      SIGNAL(.I.) := SIGNAL(.I.) * MODIFIER;
   end;

end;


(*********************************************************************)
(*                                                                   *)
(*                        procedure FREQ                             *)
(*                                                                   *)
(*   This procedure takes the discrete Fourier transform (DFT) of the *)
(*   signal given to it.  The DFT is implemented by the decimation-in-*)
(*   frequency FFT algorithm.                                        *)
(*                                                                   *)
(*********************************************************************)
procedure FREQ (     SIGNAL : SIGNAL_TYPE;
                 var FREQX : SIGNAL_TYPE);

const

   PI = 3.1415926535898;
   MAX_ARRAY_SIZE = 2048;

var

   DUMMY,
   I,
   J,
   M,
   N,
   L,
   NV2,
   NM1,
   IP,
   LE,
   K      : INTEGER;
   LARGEST,
   TEST,
   LE1,
   UR,
   UI,
   WR,
   WI,
   TR,
   TI,
   TMR,
   TMI      : REAL;
   MAG,
   PHASE,
   XR,
   XI       : array(.1..MAX_ARRAY_SIZE.) of REAL;

(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
(*                                                               *)
(*                     function RAISE                            *)
(*                                                               *)
(*   This function raises A to the integer power N.  This function is *)
(*   not intrinsically available in the Pascal compiler.         *)
(*                                                               *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
function RAISE ( A,N : INTEGER ) : INTEGER;

var

   TEMP,
   I : INTEGER;

begin

   TEMP := 1;
   for I := 1 to N do
      TEMP := TEMP * A;
   RAISE := TEMP;

end;


begin                                   (*          of FREQ          *)

   N := SIGNAL_SIZE;
   for I := 1 to N do
   begin
      XR(.I.) := SIGNAL(.I.);
      XI(.I.) := 0;
   end;
   M := 0;                              (* calculate M               *)
   TEST := N;
   repeat
      TEST := TEST/2;
      M := M + 1;
   until TEST <= 1;
   if TEST < 1 then
      for I := N+1 to RAISE(2,M) do     (* length /=2**M, pad with 0 *)
         XR(.I.) := 0.0;
   N := RAISE(2,M);
   for L := 1 to M do                   (* calculate FFT             *)
   begin
      LE := RAISE(2,M+1-L);
      LE1 := LE / 2.0;
      UR := 1;
      UI := 0;
      WR := cos(PI/LE1);
      WI := -sin(PI/LE1);
      for J := 1 to round(LE1) do
      begin
         for K := round((J-1)/LE) to round(N/LE) do
         begin
            I := K*LE + J;
            if I <= N then
            begin
               IP := I + round(LE1);
               TR := XR(.I.) + XR(.IP.);
               TI := XI(.I.) + XI(.IP.);
               TMR :=XR(.I.) - XR(.IP.);
               TMI := XI(.I.) - XI(.IP.);
               XR(.IP.) := TMR*UR - TMI*UI;
               XI(.IP.) := TMR*UI + TMI*UR;
               XR(.I.) := TR;
               XI(.I.) := TI;
            end;
         end; (* NEXT K *)
         TR := UR*WR - UI*WI;
         UI := UR*WI + UI*WR;
         UR := TR;
      end; (* NEXT J *)
   end; (* NEXT L *)
   NV2 := N div 2;
   NM1 := N-1;
   J := 1;
   for I := 1 to NM1 do
   begin
      if I < J then
      begin
         TR := XR(.I.);
         XR(.I.) := XR(.J.);
         XR(.J.) := TR;
         TI := XI(.I.);
         XI(.I.) := XI(.J.);
         XI(.J.) := TI;
      end;
      K := NV2;
      while K < J do
      begin
         J := J-K;
         K := K div 2;
      end;
      J := J+K;
   end; (* NEXT I *)

   for I := 1 to N do                     (* calc mag & phase of DFT   *)
   begin
      MAG(.I.) := sqrt(sqr(XR(.I.)) + sqr(XI(.I.)));
      PHASE(.I.) := 0.0;
      if XR(.I.) <> 0.0 then
         PHASE(.I.) := arctan(XI(.I.)/XR(.I.)) * 180 / PI;
      if XR(.I.) < 0.0 then
         if XI(.I.) > 0.0 then
            PHASE(.I.) := 180 + PHASE(.I.)
         else
            PHASE(.I.) := -180 + PHASE(.I.)
   end;

   for I := 1 to SIGNAL_SIZE do
       FREQX(.I.) := MAG(.I.);

end;                                      (*         of FREQ          *)


(****************************************************************)
(*                                                              *)
(*                      procedure FILTER                        *)
(*                                                              *)
(*    Purpose :  This program was designed to implement a generalized*)
(*             digital filter according to the following scheme.  *)
(*                                                              *)
(*         Once a window size has been chosen the user is asked to  *)
(*         supply the requested weighting values or choose a       *)
(*          predefined set for either the time or rank weights.          *)
(*                                                                        *)
(*                                                                        *)
(*                                                                        *)
(*             time ordered  --------------------                         *)
(*                input -->  |    |    |  ...   |  |                      *)
(*                           --------------------                         *)
(*                              ↑    ↑    ↑...    ↑                       *)
(*                         A(1)-0   -0   -0...A(n)-0                      *)
(*                              ↑    ↑    ↑       ↑                       *)
(*                           --------------------                         *)
(*                           |        SORT        |                       *)
(*                           --------------------                         *)
(*                              ↑    ↑    ↑...    ↑                       *)
(*                         B(1)-0   -0   -0...B(n)-0                      *)
(*                              ↑    ↑    ↑       ↑                       *)
(*                           --------------------                         *)
(*                           |   +    +   ..+.. +  | = output             *)
(*                           --------------------                         *)
(*                                                                        *)
(*              The procedure then opens the external data               *)
(*              file where the values of the sampled                      *)
(*              signal to be filtered are and initializes the window.    *)
(*              Initializing consists of filling the first half of the*)
(*              window with the first data value and then reading in     *)
(*              subsequent values until the window is filled.  Next      *)
(*              the window is weighted by the W() set of coefficients    *)
(*              The window is then rank ordered.  After this the         *)
(*              second set of coefficients are applied which trim off    *)
(*              the smallest and largest values of the data.             *)
(*              After this last trimming the remaining samples are       *)
(*              summed and gain adjusted.  Finally, the                   *)
(*              window is stepped over one value in time by deleting     *)
(*              the oldest value and inserting the new value in the      *)
(*              proper rank order.  The filtering and stepping           *)
(*              continue until the end of the data file is reached       *)
(*                                                                        *)
(**************************************************************************)
procedure FILTER ( var SIGNAL : SIGNAL_TYPE );

const

   MAX_WINDOW_SIZE = 64;

type

   DATA_ELEMENT_POINTER = @DATA_ELEMENT;
   DATA_ELEMENT         = record
                              ACTUAL_VALUE   REAL;
                              MODIFIED_VAL   REAL;
                              NEXT_LARGEST   DATA_ELEMENT_POINTER,
                              NEXT_IN_TIME   DATA_ELEMENT_POINTER
                          end;
   WEIGHT_ARRAY_TYPE    = array( 1..MAX_WINDOW_SIZE ) of REAL.

var

   TEMP                : array( 1..1024 ) of REAL.
   SMALLEST_VALUE,
   OLDEST_VALUE        : DATA_ELEMENT_POINTER;
   I,
   WINDOW_SIZE         : INTEGER;
   FILTER_INPUT,
   FILTER_OUTPUT       : REAL;
   TIME_WEIGHT_ARRAY,
   RANK_WEIGHT_ARRAY   : WEIGHT_ARRAY_TYPE;


(*-----------------------------------------------------------------------*)
(*                                                                        *)
(*                    procedure GET_PARAMETERS                            *)
(*                                                                        *)
(*     This procedure returns the user input parameters that will be  *)
(*     used during the program run.                                     *)
(*                                                                      *)
(*----------------------------------------------------------------------*)
procedure GET_PARAMETERS ( var WINDOW_SIZE : INTEGER;
                           var TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                           var RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE);

var

   ANS,
   CHOICE,
   I                  : INTEGER;
   CHECK,
   VALID_SIZE         : BOOLEAN;
   GAIN,
   DUMMY              : REAL;
   COEFF_FILE         : TEXT;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                      *)
(*                       procedure MANUAL_INPUT                         *)
(*                                                                      *)
(*     This procedure lets the user input the weighting                 *)
(*     coefficients to be used.                                         *)
(*                                                                      *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
   procedure MANUAL_INPUT( var INPUT_ARRAY : WEIGHT_ARRAY_TYPE;
                               WINDOW_SIZE : INTEGER);

   var

      I,
      ANS        INTEGER;

   begin

      writeln( Input the desired filter coefficients.');
      for I  = 1 to round(WINDOW_SIZE/2) do
      begin
         writeln( A( ,I 2, ) = ?');
         readln(INPUT_ARRAY( I ));
      end.
      writeln( Is the filter symmetric (1-yes,0-no)?');
      readln(ANS).
      for I  = round(WINDOW_SIZE/2)+1 to WINDOW_SIZE do
         if ANS = 1 then
            INPUT_ARRAY( I )  =INPUT_ARRAY( WINDOW_SIZE - I + 1.)
         else
         begin
            writeln( A( ,I 2, ) = ?').
            readln(INPUT_ARRAY( I )).
         end

   end


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                      *)
(*                       procedure DO_ALPHA_TRIM                        *)
(*                                                                      *)
(*     This procedure calculates the number of elements to be          *)
(*     trimmed off of each end of the rank weight array and 0's them.  *)
(*                                                                      *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
   procedure DO_ALPHA_TRIM ( var RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                                 WINDOW_SIZE   INTEGER);

   var

      I,
      NUM_TRIMMED,
      TRIM,
      NUM_LEFT     : INTEGER;

   begin

      writeln('Input how many elements to trim (must be even #))');
      readln(NUM_TRIMMED);
      TRIM := NUM_TRIMMED div 2;
      NUM_LEFT := WINDOW_SIZE - NUM_TRIMMED;
      for I := 1 to TRIM do
         RANK_WEIGHT_ARRAY(.I.) := 0;
      for I := TRIM+1 to TRIM+NUM_LEFT do
         RANK_WEIGHT_ARRAY(.I.) := 1;
      for I := TRIM+NUM_LEFT+1 to WINDOW_SIZE do
         RANK_WEIGHT_ARRAY(.I.) := 0;

   end;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                   *)
(*                     procedure CALCULATE                           *)
(*                                                                   *)
(*    This procedure calculates the gain factor for the filter       *)
(*    coefficients so that the output is properly scaled.            *)
(*                                                                   *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
   procedure CALCULATE ( var GAIN : REAL;
                             WINDOW_SIZE : INTEGER;
                             TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                             RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE);

   var

      A,
      J,
      I     : INTEGER;
      ORDER : SIGNAL_TYPE;

   begin

   for I := 1 to WINDOW_SIZE do
      ORDER(.I.) := TIME_WEIGHT_ARRAY(.I.);
   SORT(WINDOW_SIZE,ORDER);
   GAIN := 0.0;
   for I := 1 to WINDOW_SIZE do
      GAIN := GAIN + ORDER(.I.) * RANK_WEIGHT_ARRAY(.I.);

   end;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                   *)
(*                     procedure SHOW                                *)
(*                                                                   *)
(*    This procedure clears the screen and then provides a list      *)
(*    of the coefficients to the user.                               *)
(*                                                                   *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
   procedure SHOW ( TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                    RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE;
                    WINDOW_SIZE : INTEGER);

   var

      I     INTEGER;

   begin
      CMS('CLRSCRN',I);
      writeln('TIME COEFF       VALUE          RANK COEFF       VALUE');
      for I  = 1 to WINDOW_SIZE do
```

```
          writeln('  A(',I:2,') =',TIME_WEIGHT_ARRAY(.I.):12:8,
           '        B(',I:2,') =',RANK_WEIGHT_ARRAY(.I.):12:8);
    end;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                        *)
(*                        procedure MODIFY                                *)
(*                                                                        *)
(*     This procedure modifies either the whole weight array or           *)
(*     individual coefficient values that the user desires.               *)
(*                                                                        *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
    procedure MODIFY ( var MODIFY_ARRAY : WEIGHT_ARRAY_TYPE;
                           WINDOW_SIZE : INTEGER);

    var

       I   : INTEGER;
       VAL : REAL;

    begin
       writeln('Which element ( O-> all elements )');
       readln(I);
       if I = 0 then
         MANUAL_INPUT(MODIFY_ARRAY,WINDOW_SIZE)
       else
       begin
          writeln('What is the new value');
          readln(VAL);
          MODIFY_ARRAY(.I.) := VAL;
       end;
    end;


begin                                          (*       of GET_PARAMETERS  *)

    CMS('CLRSCRN',I);
    writeln;
    writeln('This program implements a digital filter.');
    writeln;
    repeat
     writeln('Input the desired window size(must be an odd integer).');
       writeln;
       read(WINDOW_SIZE);
       VALID_SIZE := true;
       if not(odd(WINDOW_SIZE)) then
          begin
             writeln('The window size must be an odd integer.');
             VALID_SIZE := false;
          end;
       if (WINDOW_SIZE > MAX_WINDOW_SIZE) or (WINDOW_SIZE < 0) then
          begin
             writeln('The size must be positive but less than ',
                      MAX_WINDOW_SIZE);
             VALID_SIZE := false;
          end;
    until VALID_SIZE;
    for I := 1 to WINDOW_SIZE do
    begin
       TIME_WEIGHT_ARRAY(.I.) := 0;
       RANK_WEIGHT_ARRAY(.I.) := 0;
    end;

    writeln('Would you like to initialize a predetermined filter');
    writeln('(i.e. a mean or median filter) or manually input the');
    writeln('filter coefficients?');
    writeln('                1) mean filter');
    writeln('                2) median filter');
    writeln('                3) alpha-trimmed mean filter');
    writeln('                4) alpha-trimmed linear filter');
    writeln('                5) other coefficient input');
```

```
readln(CHOICE);
CMS('CLRSCRN',I);
case CHOICE of
   1 : for I := 1 to WINDOW_SIZE do
       begin
          TIME_WEIGHT_ARRAY(.I.) := 1/WINDOW_SIZE;
          RANK_WEIGHT_ARRAY(.I.) := 1;
       end;
   2:  begin
          for I := 1 to WINDOW_SIZE do
          begin
             TIME_WEIGHT_ARRAY(.I.) := 1;
             RANK_WEIGHT_ARRAY(.I.) := 0;
          end;
          RANK_WEIGHT_ARRAY(.round(WINDOW_SIZE/2).) := 1;
       end;
   3:  begin

for I := 1 to WINDOW_SIZE do
   TIME_WEIGHT_ARRAY(.I.) := 1;

DO_ALPHA_TRIM(RANK_WEIGHT_ARRAY,WINDOW_SIZE);

CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
for I := 1 to WINDOW_SIZE do
   RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;

       end;
   4:  begin

CMS('CLRSCRN',I);
writeln;
writeln('Alpha-trimmed linear filter');
writeln;
writeln('Are the desired filter coefficients saved(1-yes,0-no)?');
readln(ANS);
if ANS = 1 then
begin
   reset(COEFF_FILE,'name=FIR.COEFF.*');
   for I := 1 to WINDOW_SIZE do
      readln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
end
else
begin
   CMS('CLRSCRN',I);
   MANUAL_INPUT(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
   writeln('Do you want to save this set (1-yes,0-no)?');
   readln(ANS);
   if ANS = 1 then
   begin
      rewrite(COEFF_FILE,'name=FIR.COEFF.*');
      for I := 1 to WINDOW_SIZE do
         writeln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
   end;
end;

CMS('CLRSCRN',I);
DO_ALPHA_TRIM(RANK_WEIGHT_ARRAY,WINDOW_SIZE);

CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
writeln('The gain for this set of coefficients is',GAIN:7:4,
        ' and the rank');
writeln('weighting coefficients have been divided by it.');
for I := 1 to WINDOW_SIZE do
   RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;

       end;
   5:  begin

writeln('Are the desired filter coefficients saved(1-yes,0-no)?');
readln(ANS);
if ANS = 1 then
```

```
   begin
      reset(COEFF_FILE,'name=FIR.COEFF.*');
      for I := 1 to WINDOW_SIZE do
         readln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
   end
   else
   begin
      CMS('CLRSCRN',I);
   writeln;
   writeln('Now input the ',WINDOW_SIZE:2,' weight parameters for the');
   writeln('time ordered buffer');
      MANUAL_INPUT(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
      writeln('Do you want to save this set (1-yes,0-no)?');
      readln(ANS);
      if ANS = 1 then
      begin
         rewrite(COEFF_FILE,'name=FIR.COEFF.*');
         for I := 1 to WINDOW_SIZE do
            writeln(COEFF_FILE,TIME_WEIGHT_ARRAY(.I.));
      end;
   end;

   CMS('CLRSCRN',I);
   writeln;
   writeln('Now the same thing for the rank ordered buffer');
   MANUAL_INPUT(RANK_WEIGHT_ARRAY,WINDOW_SIZE);

   CALCULATE(GAIN,WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   for I := 1 to WINDOW_SIZE do
      RANK_WEIGHT_ARRAY(.I.) := RANK_WEIGHT_ARRAY(.I.)/GAIN;

          end;

   end;                                    (*      of case statement    *)

   writeln('Would you like to see the coefficients(1-yes,0-no)?');
   readln(I);
   if I = 1 then
   SHOW(TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY,WINDOW_SIZE);

   writeln('Would you like to change anything(1-yes,0-no)?');
   readln(ANS);
   while ANS = 1 do
   begin
      writeln('Which set of coefficients 1->time, 2->rank');
      readln(CHOICE);
      case CHOICE of
         1 : MODIFY(TIME_WEIGHT_ARRAY,WINDOW_SIZE);
         2 : MODIFY(RANK_WEIGHT_ARRAY,WINDOW_SIZE);
      end;
      SHOW (TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY,WINDOW_SIZE);
      writeln('Change anything else(1-yes,0-no)?');
      readln(ANS);
   end;
end;                                       (*       of GET_PARAMETERS  *)


(*-------------------------------------------------------------------*)
(*                                                                   *)
(*                       procedure INITIALIZE                        *)
(*                                                                   *)
(*    This procedure initializes the buffer window by reading the    *)
(*    first data value and using it to fill the first half of the    *)
(*    window and then filling the rest of the window with the        *)
(*    subsequent data values read out of the data file.              *)
(*                                                                   *)
(*-------------------------------------------------------------------*)
procedure INITIALIZE (    WINDOW_SIZE : INTEGER;
                      var OLDEST_VALUE : DATA_ELEMENT_POINTER);
var
```

```
   ELEMENT,
   NEXT_ELEMENT : DATA_ELEMENT_POINTER;
   I,
   DUMMY,
   MIDDLE_OF_WINDOW : INTEGER;

begin                                         (*      of INITIALIZE         *)

   new(ELEMENT);                              (* allocate dynamic variable *)
   OLDEST_VALUE := ELEMENT;
   MIDDLE_OF_WINDOW := round(WINDOW_SIZE/2); (* find middle of window *)
   ELEMENT@.ACTUAL_VALUE := SIGNAL(.1-MIDDLE_OF_WINDOW+1.);

   for I := 2 to WINDOW_SIZE do                          (* read in data   *)
      begin                                   (* for last half of window   *)
         new(NEXT_ELEMENT);
         NEXT_ELEMENT@.ACTUAL_VALUE := SIGNAL(.I-MIDDLE_OF_WINDOW+1.);
         ELEMENT@.NEXT_IN_TIME := NEXT_ELEMENT;
         ELEMENT := NEXT_ELEMENT;
      end;

end;                                          (*       of INITIALIZE        *)

(*-----------------------------------------------------------------------*)
(*                                                                       *)
(*                    procedure WEIGHT_TIME_BUFFER                       *)
(*                                                                       *)
(*    This procedure weights each of the elements in the time odered    *)
(*    buffer by the user input values stored in the TIME_WEIGHT_ARRAY*)
(*                                                                       *)
(*-----------------------------------------------------------------------*)
procedure WEIGHT_TIME_BUFFER ( WINDOW_SIZE : INTEGER;
                               OLDEST_VALUE : DATA_ELEMENT_POINTER;
                               TIME_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE );

var

   I : INTEGER;
   ELEMENT : DATA_ELEMENT_POINTER;

begin

   ELEMENT := OLDEST_VALUE;
   for I := 1 to WINDOW_SIZE do
      begin
         ELEMENT@.MODIFIED_VAL := ELEMENT@.ACTUAL_VALUE *
                                  TIME_WEIGHT_ARRAY(.I.);
         ELEMENT := ELEMENT@.NEXT_IN_TIME;
      end;

end;


(*-----------------------------------------------------------------------*)
(*                                                                       *)
(*                       procedure RANK_ORDER                            *)
(*                                                                       *)
(*    This procedure takes the window that has been modified by the     *)
(*    time weights & uses an insertion sort to rank order each of the*)
(*    elements.  At the end of this procedure each element in the       *)
(*    window is ordered by size of the value as well as in time.        *)
(*                                                                       *)
(*-----------------------------------------------------------------------*)
procedure RANK_ORDER (     WINDOW_SIZE : INTEGER;
                       var OLDEST_VALUE : DATA_ELEMENT_POINTER;
                       var SMALLEST_VALUE : DATA_ELEMENT_POINTER);

var

   ELEMENT,
   NEW_ELEMENT : DATA_ELEMENT_POINTER;
   I           : INTEGER;
```

```
begin                                          (*       of RANK_ORDER        *)

   SMALLEST_VALUE := OLDEST_VALUE;             (* start with first value as *)
   SMALLEST_VALUE@.NEXT_LARGEST := nil;        (* smallest and insert each  *)
   NEW_ELEMENT := OLDEST_VALUE@.NEXT_IN_TIME; (* 1st new valu to insert*)
   for I := 2 to WINDOW_SIZE do               (* loop through each value   *)
      begin                                    (* check if new valu<smallest*)
         ELEMENT := SMALLEST_VALUE;                   (* value in order      *)
         if NEW_ELEMENT@.MODIFIED_VAL < SMALLEST_VALUE@.MODIFIED_VAL
            then begin
               NEW_ELEMENT@.NEXT_LARGEST := SMALLEST_VALUE;
               SMALLEST_VALUE := NEW_ELEMENT;
            end
         else                                  (* else find proper order    *)
            begin
               while (ELEMENT@.NEXT_LARGEST <> nil) and
                     (NEW_ELEMENT@.MODIFIED_VAL >
                      ELEMENT@.NEXT_LARGEST@.MODIFIED_VAL) do
                  ELEMENT := ELEMENT@.NEXT_LARGEST;
                                               (* and insert in place       *)
               NEW_ELEMENT@.NEXT_LARGEST := ELEMENT@.NEXT_LARGEST;
               ELEMENT@.NEXT_LARGEST := NEW_ELEMENT;
            end;
         NEW_ELEMENT := NEW_ELEMENT@.NEXT_IN_TIME; (* then go on to    *)
      end;                                     (* insert next elmnt in order*)

end;                                           (*      of RANK_ORDER        *)


(*------------------------------------------------------------------------*)
(*                                                                        *)
(*                    procedure WEIGHT_RANK_BUFFER                        *)
(*                                                                        *)
(*     This procedure weights each of the elements in the time odered *)
(*     buffer by the user input values stored in the TIME_WEIGHT_ARRAY*)
(*                                                                        *)
(*------------------------------------------------------------------------*)
procedure WEIGHT_RANK_BUFFER ( WINDOW_SIZE : INTEGER;
                               SMALLEST_VALUE : DATA_ELEMENT_POINTER;
                               RANK_WEIGHT_ARRAY : WEIGHT_ARRAY_TYPE );

var

   I : INTEGER;
   ELEMENT : DATA_ELEMENT_POINTER;

begin

   ELEMENT := SMALLEST_VALUE;
   for I := 1 to WINDOW_SIZE do
      begin
         ELEMENT@.MODIFIED_VAL := ELEMENT@.MODIFIED_VAL *
                                  RANK_WEIGHT_ARRAY(.I.);
         ELEMENT := ELEMENT@.NEXT_LARGEST;
      end;

end;


(*------------------------------------------------------------------------*)
(*                                                                        *)
(*                      procedure FILTER_M                                *)
(*                                                                        *)
(*     This procedure finds the middle value of the time sequence,       *)
(*     which is the input to the filter, and sums the values of the      *)
(*     order sequence, the filter output, and sends the two values       *)
(*     back to the main program to be stored in a graph file.            *)
(*                                                                        *)
(*------------------------------------------------------------------------*)
procedure FILTER_M    (     WINDOW_SIZE : INTEGER;
                        var OLDEST_VALUE : DATA_ELEMENT_POINTER;
```

```
                          var SMALLEST_VALUE : DATA_ELEMENT_POINTER;
                          var FILTER_INPUT : REAL;
                          var FILTER_OUTPUT : REAL);

var

   I,
   MIDDLE_OF_WINDOW : INTEGER;
   INPUT_ELEMENT,
   OUTPUT_ELEMENT   : DATA_ELEMENT_POINTER;

begin                                     (*       of FILTER_M          *)

   MIDDLE_OF_WINDOW := round(WINDOW_SIZE/2);
   INPUT_ELEMENT    := OLDEST_VALUE;
   OUTPUT_ELEMENT   := SMALLEST_VALUE;
   FILTER_OUTPUT := 0;
   for I := 1 to WINDOW_SIZE do           (* step thru window to calc  *)
      begin                               (* output                    *)
         if I < MIDDLE_OF_WINDOW then     (* filter input is middle val*)
            INPUT_ELEMENT := INPUT_ELEMENT@.NEXT_IN_TIME;
         FILTER_OUTPUT := FILTER_OUTPUT + OUTPUT_ELEMENT@.MODIFIED_VAL;
         OUTPUT_ELEMENT := OUTPUT_ELEMENT@.NEXT_LARGEST;
      end;
   FILTER_INPUT := INPUT_ELEMENT@.ACTUAL_VALUE;

end;                                      (*       of FILTER_M          *)


(*-------------------------------------------------------------------*)
(*                                                                   *)
(*                        procedure STEP_ONE                         *)
(*                                                                   *)
(*     Step the window over one data value by deleting out the oldest *)
(*     value from the time sequence and then creating                *)
(*     a new dynamic variable, reading in its new value, and         *)
(*     inserting it at the end of the time sequence.                 *)
(*                                                                   *)
(*-------------------------------------------------------------------*)
procedure STEP_ONE   (var OLDEST_VALUE : DATA_ELEMENT_POINTER;
                          NEXT_VAL : INTEGER;
                          WINDOW_SIZE : INTEGER);

var

   J,
   DUMMY : INTEGER;
   ELEMENT,
   TARGET_ELEMENT : DATA_ELEMENT_POINTER;

begin                                     (*        of STEP_ONE         *)

   (* * * *  delete oldest value from window  * * * *)

   TARGET_ELEMENT := OLDEST_VALUE;
   OLDEST_VALUE := OLDEST_VALUE@.NEXT_IN_TIME; (* delete from time seq*)

   dispose(TARGET_ELEMENT);               (* de-allocate dynamic varbl *)

   (* * * *  put new element into window  * * * *)

   new(ELEMENT);
   TARGET_ELEMENT := OLDEST_VALUE;        (* start at beginning to find*)
   for J := 1 to (WINDOW_SIZE-2) do       (* the most recent value     *)
      TARGET_ELEMENT := TARGET_ELEMENT@.NEXT_IN_TIME;
   TARGET_ELEMENT@.NEXT_IN_TIME := ELEMENT;          (* add new value *)
   ELEMENT@.NEXT_IN_TIME := nil;          (* and put nil value on end  *)
   ELEMENT@.ACTUAL_VALUE := SIGNAL(.NEXT_VAL.)


end;                                      (*        of STEP_ONE         *)
```

```
begin                                           (* of FILTER - MAIN PROGRAM  *)

   GET_PARAMETERS(WINDOW_SIZE,TIME_WEIGHT_ARRAY,RANK_WEIGHT_ARRAY);
   INITIALIZE(WINDOW_SIZE,OLDEST_VALUE);
   for I := 1 to SIGNAL_SIZE do
   begin
      WEIGHT_TIME_BUFFER(WINDOW_SIZE,OLDEST_VALUE,TIME_WEIGHT_ARRAY);
      RANK_ORDER(WINDOW_SIZE,OLDEST_VALUE,SMALLEST_VALUE);
      WEIGHT_RANK_BUFFER(WINDOW_SIZE,SMALLEST_VALUE,RANK_WEIGHT_ARRAY);
      FILTER_M(WINDOW_SIZE,OLDEST_VALUE,SMALLEST_VALUE,FILTER_INPUT,
            FILTER_OUTPUT);
      TEMP(.I.) := FILTER_OUTPUT;
      if I <> SIGNAL_SIZE then
         STEP_ONE(OLDEST_VALUE,(I+round(WINDOW_SIZE/2)),WINDOW_SIZE);
   end;
   for I := 1 to SIGNAL_SIZE do
      SIGNAL(.I.) := TEMP(.I.);

end;                                            (* of FILTER - MAIN PROGRAM  *)


(*********************************************************************)
(*                                                                   *)
(*                     procedure TRANS_AND_STORE                     *)
(*                                                                   *)
(*   This procedure takes the transfer function of the filter by    *)
(*   dividing the magnitude of the DFT of the output of the filter  *)
(*   by the magnitude of the DFT of the input to the filter. This   *)
(*   result is then stored for the number of trials attempted so    *)
(*   that the individual transfer function of each trial is         *)
(*   remembered for further processing.                             *)
(*                                                                   *)
(*********************************************************************)
procedure TRANS_AND_STORE (      FREQX, FREQY : SIGNAL_TYPE;
                             var Tyx_SUM : MEDIAN_TYPE;
                                 TRIAL : INTEGER );

var

   I      : INTEGER;
   FREQT  : REAL;


begin

   for I := 1 to SIGNAL_SIZE div 2 do
   begin
      FREQT := FREQY(.I.)/FREQX(.I.);
      Tyx_SUM(.I,TRIAL.) := Tyx_SUM(.I,TRIAL.) + FREQT;
   end;

end;


(*********************************************************************)
(*                                                                   *)
(*                     procedure MEDIAN_AND_PRINT                    *)
(*                                                                   *)
(*   This procedure sorts the data of the previously calculated     *)
(*   transfer functions on a point by point basis.  The median value *)
(*   of each point is selected as the output of the filter and stored *)
(*   in a file for further graphing.                                *)
(*                                                                   *)
(*********************************************************************)
procedure MEDIAN_AND_PRINT ( ORDER_ARRAY : MEDIAN_TYPE;
                             TRIALS  : INTEGER );

var
```

```
   Tyx      : SIGNAL_TYPE;
   ANS,
   NUM,
   INDEX,
   INDEX2,
   INC,
   T_INDEX,
   J,
   I          : INTEGER;
   T_VALUE,
   LARGEST_T,
   NORM_T     : REAL;
   FILET      : TEXT;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
(*                                                                  *)
(*                         function LOG                             *)
(*                                                                  *)
(*   This function calculates the base 10 logarithm of X.  The Pascal *)
(*   compiler only provides the natural logarithm as an intrinsic    *)
(*   function. The base 10 logarithm is calculated by                *)
(*                                                                  *)
(*                 log X = ln X                                     *)
(*                         -----                                    *)
(*                         ln 10                                    *)
(*                                                                  *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
function LOG (X:REAL):REAL;

begin
   if X = 0.0 then
      X := 0.00000001;
   LOG := ln(X)/ln(10.0);
end;


begin                                       (*  of MEDIAN_AND_PRINT      *)

   INDEX := SIGNAL_SIZE div 2;
   INDEX2 := SIGNAL_SIZE;
   for I := 1 to INDEX do                   (* sort each point of Tyx    *)
   begin
      for J := 1 to TRIALS do
         Tyx(.J.) := ORDER_ARRAY(.I,J.);
      SORT(TRIALS,Tyx);
      for J := 1 to TRIALS do
         ORDER_ARRAY(.I,J.) := Tyx(.J.);
   end;
   for I := 1 to INDEX do                   (*  choose median as response*)
      Tyx(.I.) := ORDER_ARRAY(.I,round(TRIALS/2).);
   readln(ANS);
   if ANS = 1 then
   begin
      LARGEST_T := Tyx(.1.);
      T_VALUE := LARGEST_T;
      T_INDEX := 1;
      for I := 1 to INDEX do                (* find largest value of Tyx *)
      begin                                 (* for normalization         *)
         LARGEST_T := max(LARGEST_T,Tyx(.I.));
         if LARGEST_T <> T_VALUE then
         begin
            T_INDEX := I;
            T_VALUE := LARGEST_T;
         end;
      end;
      NORM_T := 0;
      INC := round(INDEX * 0.02);           (* choose interval around    *)
      NUM := 0;                             (* largest value for normaliz*)
      for I := max(1,T_INDEX-INC) to min(T_INDEX+INC,INDEX) do
      begin                                 (* calculate norm factor     *)
         NORM_T := NORM_T + Tyx(.I.);
         NUM := NUM + 1;
      end;
      NORM_T := NORM_T/NUM;
      for I := 1 to INDEX do                 (* normalize Tyx               *)
         Tyx(.I.) := 20*LOG(Tyx(.I.)/NORM_T);
   end;

   rewrite(FILET,'name=MEDTRANS.PDATA.*');
   for I := 1 to INDEX do
      writeln(FILET,(I/INDEX2):10:8,' ',Tyx(.I.));

end;


(**************************************************************************)
(*                                                                        *)
(*                       procedure AVERAGE_AND_PRINT                      *)
(*                                                                        *)
(*   This procedure averages the data of the previously calculated        *)
(*   transfer functions on a point by point basis.  The average value     *)
(*   at each point is calculated  the output of the filter and stored     *)
(*   in a file for further graphing.                                      *)
(*                                                                        *)
(**************************************************************************)
procedure AVERAGE_AND_PRINT ( ARR : MEDIAN_TYPE;
                              TRIALS : INTEGER );

var

   Tyx_S : SIGNAL_TYPE;
   ANS,
   NUM,
   INDEX,
   INDEX2,
   INC,
   T_INDEX,
   J,
   I          : INTEGER;
   T_VALUE,
   LARGEST_T,
   NORM_T     : REAL;
   FILET      : TEXT;


(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
(*                                                                        *)
(*                           function LOG                                 *)
(*                                                                        *)
(*   This function calculates the base 10 logarithm of X.  The Pascal     *)
(*   compiler only provides the natural logarithm as an intrinsic         *)
(*   function. The base 10 logarithm is calculated by                     *)
(*                                                                        *)
(*                 log X = ln X                                           *)
(*                         -----                                          *)
(*                         ln 10                                          *)
(*                                                                        *)
(*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*)
function LOG (X:REAL):REAL;

begin
   if X = 0.0 then
      X := 0.00000001;
   LOG := ln(X)/ln(10.0);
end;


begin                                       (*  of AVERAGE_AND_PRINT     *)

   INDEX := SIGNAL_SIZE div 2;
   INDEX2 := SIGNAL_SIZE;
   for I := 1 to INDEX do
   begin                                      (* average each point to     *)
      Tyx_S(.I.) := 0.0;                      (* calculate Tyx             *)
      for J := 1 to TRIALS do
         Tyx_S(.I.) := Tyx_S(.I.) + ARR(.I,J.);
   end;
   readln(ANS);
   if ANS = 1 then
   begin                                      (* choose largest value of   *)
      LARGEST_T := Tyx_S(.1.);                (* Tyx to normalize          *)
      T_VALUE := LARGEST_T;
      T_INDEX := 1;
      for I := 1 to INDEX do
      begin
         LARGEST_T := max(LARGEST_T,Tyx_S(.I.));
         if LARGEST_T <> T_VALUE then
         begin
            T_INDEX := I;
            T_VALUE := LARGEST_T;
         end;
      end;
      NORM_T := 0;
      INC := round(INDEX * 0.02);             (* calculate normalization   *)
      NUM := 0;                               (* factor over range near max*)
      for I := max(1,T_INDEX-INC) to min(T_INDEX+INC,INDEX) do
      begin
         NORM_T := NORM_T + Tyx_S(.I.);
         NUM := NUM + 1;
      end;
      NORM_T := NORM_T/NUM;
      for I := 1 to INDEX do                  (* normalize Tyx             *)
         Tyx_S(.I.) := 20*LOG(Tyx_S(.I.)/NORM_T);
   end
   else
      for I := 1 to INDEX do
         Tyx_S(.I.) := Tyx_S(.I.)/TRIALS;

   rewrite(FILET,'name=SUMTRANS.PDATA.*');
   for I := 1 to INDEX do
      writeln(FILET,(I/INDEX2):10:8,' ',Tyx_S(.I.));

end;


begin                                         (*   of MEGAMEDIAN           *)

   termout(OUTPUT);
   reset(INPUT,'name=TRIALS.MEGA.B');
   readln(TRIALS);
   datetime(DATE,TIME);
   readstr(str(TIME),I:2,CH,SEED1:2,CH,SEED2);
   I := I*SEED1*SEED2+I+SEED1+SEED2;
   DUMMY := random(I);
   SIGNAL_SIZE := 256;
   for I := 1 to TRIALS do
   begin
      reset(INPUT,'name=INDATA.MEGA.B');
      MAKEDATA(SIGNAL);
      WINDOW(SIGNAL);
      FREQ(SIGNAL,FFTX_MAG);
      FILTER(SIGNAL);
      FREQ(SIGNAL,FFTY_MAG);
      TRANS_AND_STORE(FFTX_MAG,FFTY_MAG,Tyx_SUM,I);
   end;
   MEDIAN_AND_PRINT(Tyx_SUM,TRIALS);
   AVERAGE_AND_PRINT(Tyx_SUM,TRIALS);

end.                                          (*   of MEGAMEDIAN           *)
```

END
DATE
FILMED
JAN
1988